ProSoft TECHNOLOGY

Where Automation Connects.

MVI56E-MCM

ControlLogix Platform

Modbus Communication Module

June 17, 2014

USER MANUAL

Your Feedback Please

We always want you to feel that you made the right decision to use our products. If you have suggestions, comments, compliments or complaints about our products, documentation, or support, please write or call us.

ProSoft Technology
5201 Truxtun Ave., 3rd Floor
Bakersfield, CA 93309
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
support@prosoft-technology.com

© 2014 ProSoft Technology, Inc. All rights reserved.

MVI56E-MCM User Manual

June 17, 2014

ProSoft Technology®, is a registered copyright of ProSoft Technology, Inc. All other brand or product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

In an effort to conserve paper, ProSoft Technology no longer includes printed manuals with our product shipments. User Manuals, Datasheets, Sample Ladder Files, and Configuration Files are provided on the enclosed DVD and are available at no charge from our web site: http://www.prosoft-technology.com

Content Disclaimer

This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither ProSoft Technology nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. Information in this document including illustrations, specifications and dimensions may contain technical inaccuracies or typographical errors. ProSoft Technology makes no warranty or representation as to its accuracy and assumes no liability for and reserves the right to correct such inaccuracies or errors at any time without notice. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of ProSoft Technology. All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components. When devices are used for applications with technical safety requirements, the relevant instructions must be followed. Failure to use ProSoft Technology software or approved software with our hardware products may result in injury, harm, or improper operating results. Failure to observe this information can result in injury or equipment damage.

© 2014 ProSoft Technology. All Rights Reserved.

Printed documentation is available for purchase. Contact ProSoft Technology for pricing and availability.

North America: +1 (661) 716-5100
Asia Pacific: +603.7724.2080
Europe, Middle East, Africa: +33 (0) 5.3436.87.20
Latin America: +1.281.298.9109

Important Safety Information

North America Warnings

A Warning - Explosion Hazard - Substitution of components may impair suitability for Class I, Division 2.
B Warning - Explosion Hazard - When in Hazardous Locations, turn off power before replacing or rewiring modules.
C Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be nonhazardous.
D Suitable for use in Class I, Division 2 Groups A, B, C, and D, Hazardous Locations or Non-Hazardous Locations.

ATEX/IECEx Warnings and Conditions of Safe Usage:

Power, Input, and Output (I/O) wiring must be in accordance with the authority having jurisdiction

A Warning - Explosion Hazard - When in hazardous locations, turn off power before replacing or wiring modules.
B Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.
C These products are intended to be mounted in an ATEX/IECEx Certified, tool-secured, IP54 enclosure. The devices shall provide external means to prevent the rated voltage being exceeded by transient disturbances of more than 40%. This device must be used only with ATEX certified backplanes.
D Before operating the reset switch, be sure the area is known to be non-hazardous.

Agency Approvals and Certifications

Agency
RoHS
ATEX
CSA
CE
CSA CB Safety
cULus
GOST-R
Lloyds

<Ex>
II 3 G
Ex nA IIC T4 Gc
0°C <= Ta <= 60°C
-25°C <= Ta <= 70°C (XT models only)
II – Equipment intended for above ground use (not for use in mines).
3 – Category 3 equipment, investigated for normal operation only.
G – Equipment protected against explosive gasses. <cULus>
E183151
Class I, DIV 2, groups A,B,C,D
T5 for all models
0°C to +60°C
-25°C to +70°C (XT models only)

Battery Life Advisory

Note: Modules manufactured after April 1st, 2011 do not contain a battery. For modules manufactured before that date the following applies:

The module uses a rechargeable Lithium Vanadium Pentoxide battery to back up the real-time clock and CMOS settings. The battery itself should last for the life of the module. However, if left in an unpowered state for 14 to 21 days, the battery may become fully discharged and require recharging by being placed in a powered-up ControlLogix chassis. The time required to fully recharge the battery may be as long as 24 hours.

Once it is fully charged, the battery provides backup power for the CMOS setup and the real-time clock for approximately 21 days. Before you remove a module from its power source, ensure that the battery within the module is fully charged (the BATT LED on the front of the module goes OFF when the battery is fully charged). If the battery is allowed to become fully discharged, the module will revert to the default BIOS and clock settings.

Note: The battery is not user-replaceable or serviceable.

Contents

Your Feedback Please 2
Content Disclaimer 2
Important Safety Information 3
Battery Life Advisory 4

1 Start Here 9

1.1 System Requirements 9
1.2 Deployment Checklist 10
1.3 Package Contents 12
1.4 Setting Jumpers 12
1.5 Installing the Module in the Rack 13
1.6 Creating a New RSLogix 5000 Project 14
1.6.1 Before You Import the Add-On Instruction 15
1.6.2 Creating the Module 16
1.6.3 Import the Ladder Rung 19
1.6.4 Adding Multiple Modules (Optional) 22
1.6.5 Adjust the Input and Output Array Sizes (Optional) 27
1.7 Connecting Your PC to the ControlLogix Processor 28
1.8 Downloading the Sample Program to the Processor 30
1.8.1 Configuring the RSLinx Driver for the PC COM Port 31

2 Configuration as a Modbus Master 33

2.1 Overview 33
2.2 ModDef Settings 34
2.2.1 Port Configuration 35
2.2.2 Master Command Configuration 37
2.2.3 Other Modbus Addressing Schemes 40
2.3 Master Command Examples 41
2.3.1 Read Holding Registers 4x (Modbus Function Code 3) 41
2.3.2 Read Input Registers 3x (Modbus Function Code 4) 42
2.3.3 Read Coil Status 0x (Modbus Function Code 1) 42
2.3.4 Read Input Status 1x (Modbus Function Code 2) 43
2.3.5 Force (Write) Single Coil 0x (Modbus Function Code 5) 44
2.3.6 Force (Write) Multiple Coils 0x (Modbus Function Code 15) 44
2.3.7 Preset (Write) Single Register 4x (Modbus Function Code 6) 45
2.3.8 Preset (Write) Multiple Registers 4x (Modbus Function Code 16) 46
2.4 Floating-Point Data Handling (Modbus Master) 46
2.4.1 Read Floating-Point Data 47
2.4.2 Read Multiple Floating-Point Registers 48
2.4.3 Write Floats to Slave Device 49
2.4.4 Read Floats with Single Modbus Register Address (Enron/Daniel Float) 50
2.4.5 Write to Enron/Daniel Floats 51
2.5 Command Control and Event Command 52
2.5.1 Command Control 53
2.5.2 Event Command 54

3 Configuration as a Modbus Slave 57

3.1 Overview ... 57
3.2 ModDef Settings ... 58
3.2.1 Modbus Memory Map ... 59
3.2.2 Customizing the Memory Map ... 61
3.3 Slave Configuration ... 62
3.4 Floating-Point Data Handling (Modbus Slave) ... 63
3.4.1 Enron/Daniel Float Configuration ... 64
3.5 Read and Write Same Modbus Address (Pass Through) ... 65

4 Verify Communication 67

4.1 Verifying Master Communications ... 67
4.1.1 MVI56E-MCM Status Data Definition as a Master ... 67
4.1.2 Command Error Codes ... 69
4.1.3 MCM Status Data ... 72
4.2 Verify Slave Communications ... 73
4.2.1 MVI56E-MCM Status Data Definition as a Slave ... 73

5 Diagnostics and Troubleshooting 75

5.1 Ethernet LED Indicators ... 75
5.1.1 Scrolling LED Status Indicators ... 76
5.1.2 Non-Scrolling LED Status Indicators ... 77
5.2 Clearing a Fault Condition ... 77
5.3 Troubleshooting the LEDs ... 78
5.4 Setting Up ProSoft Configuration Builder ... 78
5.4.1 Installing ProSoft Configuration Builder ... 78
5.4.2 Setting Up the Project ... 79
5.4.3 Assigning an IP Address in the Project ... 81
5.5 Connecting Your PC to the Module ... 83
5.5.1 Download the IP Address through CIPconnect ... 83
5.5.2 Using RSWho to Connect to the Module ... 92
5.5.3 Connecting Your PC to the Module's Ethernet Port ... 94
5.6 Downloading the Project to the Module ... 98
5.7 Using the Diagnostics Menu in ProSoft Configuration Builder ... 99
5.7.1 The Diagnostics Menu ... 103
5.7.2 Monitoring Backplane Information ... 103
5.7.3 Monitoring Database Information ... 104
5.7.4 Monitoring General Information ... 105
5.7.5 Monitoring Modbus Port Information ... 105
5.7.6 Data Analyzer ... 107
5.8 Reading Status Data from the Module ... 111
5.8.1 Viewing the Error Status Table ... 111
5.9 Configuration Error Codes ... 111
5.10 Connect to the Module's Web Page ... 113

6 Reference 115

6.1 Product Specifications ... 115
6.1.1 General Specifications ... 115
6.1.2 General Specifications - Modbus Master/Slave ... 116

6.1.3 Functional Specifications 116
6.1.4 Hardware Specifications 117
6.2 Functional Overview 118
6.2.1 About the Modbus Protocol 118
6.2.2 Backplane Data Transfer 118
6.2.3 Normal Data Transfer 120
6.2.4 Special Function Blocks 122
6.2.5 Data Flow Between MVI56E-MCM Module and ControlLogix Processor 134
6.3 Cable Connections 138
6.3.1 Ethernet Cable Specifications 138
6.3.2 Ethernet Cable Configuration 139
6.3.3 Ethernet Performance 139
6.3.4 RS-232 Application Port(s) 140
6.3.5 RS-422 142
6.3.6 RS-485 Application Port(s) 142
6.3.7 DB9 to RJ45 Adaptor (Cable 14) 143
6.4 MVI56E-MCM Database Definition 143
6.5 MVI56E-MCM Configuration Data 144
6.5.1 Backplane Setup 144
6.5.2 Port 1 Setup 145
6.5.3 Port 2 Setup 147
6.5.4 Port 1 Commands 150
6.5.5 Port 2 Commands 150
6.5.6 Misc. Status 150
6.5.7 Command Control 152
6.6 MVI56E-MCM Status Data Definition 152
6.7 MVI56E-MCM User Defined Data Types 154
6.7.1 MCMModuleDef 154
6.7.2 MCMCONFIG 154
6.7.3 MCMDATA 157
6.7.4 MCMSTATUS 157
6.7.5 MCMCONTROL 158
6.7.6 Util 158
6.8 Modbus Protocol Specification 160
6.8.1 Commands Supported by the Module 160
6.8.2 Read Coil Status (Function Code 01) 161
6.8.3 Read Input Status (Function Code 02) 162
6.8.4 Read Holding Registers (Function Code 03) 163
6.8.5 Read Input Registers (Function Code 04) 164
6.8.6 Force Single Coil (Function Code 05) 165
6.8.7 Preset Single Register (Function Code 06) 166
6.8.8 Diagnostics (Function Code 08) 167
6.8.9 Force Multiple Coils (Function Code 15) 169
6.8.10 Preset Multiple Registers (Function Code 16) 170
6.8.11 Modbus Exception Responses 170
6.9 Using the Optional Add-On Instruction 172
6.9.1 Before You Begin 172
6.9.2 Overview 172
6.9.3 Importing the Utility Add-On Instruction 173
6.9.4 Reading the Ethernet Settings from the Module 176
6.9.5 Writing the Ethernet Settings to the Module 177
6.9.6 Reading the Clock Value from the Module 179
6.9.7 Writing the Clock Value to the Module 179
6.10 Using the Sample Program - RSLogix 5000 Version 15 and earlier 180

6.10.1 Using the Sample Program in a New Application 181
6.10.2 Using the Sample Program in an Existing Application 186

7 Support, Service & Warranty 195

7.1 Contacting Technical Support 195
7.2 Warranty Information 196

Index 197

1 Start Here

In This Chapter

- System Requirements 9
- Deployment Checklist 10
- Package Contents 12
- Setting Jumpers 12
- Installing the Module in the Rack 13
- Creating a New RSLogix 5000 Project 14
- Connecting Your PC to the ControlLogix Processor 28
- Downloading the Sample Program to the Processor 30

To get the most benefit from this User Manual, you should have the following skills:

- **Rockwell Automation® RSLogix™ software:** launch the program, configure ladder logic, and transfer the ladder logic to the processor
- **Microsoft Windows®:** install and launch programs, execute menu commands, navigate dialog boxes, and enter data
- **Hardware installation and wiring:** install the module, and safely connect Modbus and ControlLogix devices to a power source and to the MVI56E-MCM module's application port(s)

1.1 System Requirements

The MVI56E-MCM module requires the following minimum hardware and software components:

- Rockwell Automation ControlLogix® processor (firmware version 10 or higher) with compatible limited voltage power supply and one free slot in the rack for the MVI56E-MCM module. The module requires 800mA of available 5 VDC and 3 mA of available 24 VDC power.

- Rockwell Automation RSLogix 5000 programming software
  - Version 16 or higher required for Add-On Instruction
  - Version 15 or lower must use Sample Ladder, available from www.prosoft-technology.com
- Rockwell Automation RSLinx® communication software version 2.51 or higher
- ProSoft Configuration Builder (PCB) (included)

- Pentium® II 450 MHz minimum. Pentium III 733 MHz (or better) recommended
- Supported operating systems:
  - Microsoft Windows® Vista
  - Microsoft Windows XP Professional with Service Pack 1 or 2
  - Microsoft Windows 7 Professional (32-or 64-bit)
  - Microsoft Windows 2000 Professional with Service Pack 1, 2, or 3
  - Microsoft Windows Server 2003
- 128 Mbytes of RAM minimum, 256 Mbytes of RAM recommended
- 100 Mbytes of free hard disk space (or more based on application requirements)
- 256-color VGA graphics adapter, 800 x 600 minimum resolution (True Color 1024 × 768 recommended)
- DVD drive

Note: The Hardware and Operating System requirements in this list are the minimum recommended to install and run software provided by ProSoft Technology®. Other third party applications may have different minimum requirements. Refer to the documentation for any third party applications for system requirements.

Note: You can install the module in a local or remote rack. For remote rack installation, the module requires EtherNet/IP or ControlNet communication with the processor.

1.2 Deployment Checklist

Before you begin configuring the module, consider the following questions. Your answers will help you determine the scope of your project, and the configuration requirements for a successful deployment.

1 _____ Are you creating a new application or integrating the module into an existing application?

Most applications can use the Sample Add-On Instruction or Sample Ladder Logic without any edits to the Sample Program.

2 _____ Which slot number in the chassis will the MVI56E-MCM module occupy?

For communication to occur, you must enter the correct slot number in the sample program.

3 _____ Are RSLogix 5000 and RSLinx software installed?

RSLogix and RSLinx are required to communicate to the ControlLogix processor (1756-L1, L55, L61 & L63). Sample Ladder programs are available for different versions of RSLogix 5000.

4 _____ How many words of data do you need to transfer in your application (from ControlLogix to Module / to ControlLogix from Module)?

The MVI56E-MCM module can transfer a maximum of 10,000 (16-bit) registers to and from the ControlLogix processor. The Sample Ladder transfers 600 words to the ControlLogix processor (into the Read Data array), and obtains 600 words from the ControlLogix processor (from the Write Data array)

5 _____ Will you be using the module as a Modbus Master or Modbus Slave? Will you be transferring data using Modbus RTU or Modbus ASCII?

Modbus is a Master/Slave network. Only one Master is allowed on the serial communications line (max 32 devices/RS485). The Master is responsible for polling data from the Slaves on the network.

6 _____ For a Modbus Master, what Slave Device Addresses and Modbus Data Addresses do you need to exchange data with on the Modbus network?

For a Modbus Master, you must know the Slave Device Address number of each Slave device to poll. You also need the Modbus address (for example, coil 00001, register 40001) of the data to read from or write to each Slave device.

7 _____ For a Modbus Slave, how many words or bits of data do you need to send to the Master device?

The MVI56E-MCM module can send data to a Modbus Master as 0x coil data, 1x input coil data, 3x input registers, and 4x holding registers. The sample program transfers 600 (16-bit) words or 9600 bits to the ControlLogix processor, and 600 (16-bit) words or 9600 bits from the ControlLogix processor.

8 Serial Communication Parameters for the Modbus network:

_____ Baud rate?
_____ Data bits?
_____ Parity?
_____ Stop bits?

Required for both Master and Slave configurations.

9 _____ Wiring type to use (RS232, 422 or 485). Configured by Setting Jumpers.

Required for proper implementation of the module in Master and Slave configurations.

Note: If you are using RSLogix 5000 version 16 or newer, refer to **Before You Import the Add-On Instruction (page 15).**
For RSLogix 5000 version 15 and earlier, refer to Using the Sample Program - RSLogix 5000 Version 15 and earlier (page 180).

Most applications can use the Sample Ladder Logic without modification.

1.3 Package Contents

The following components are included with your MVI56E-MCM module, and are all required for installation and configuration.

Important: Before beginning the installation, please verify that all of the following items are present.

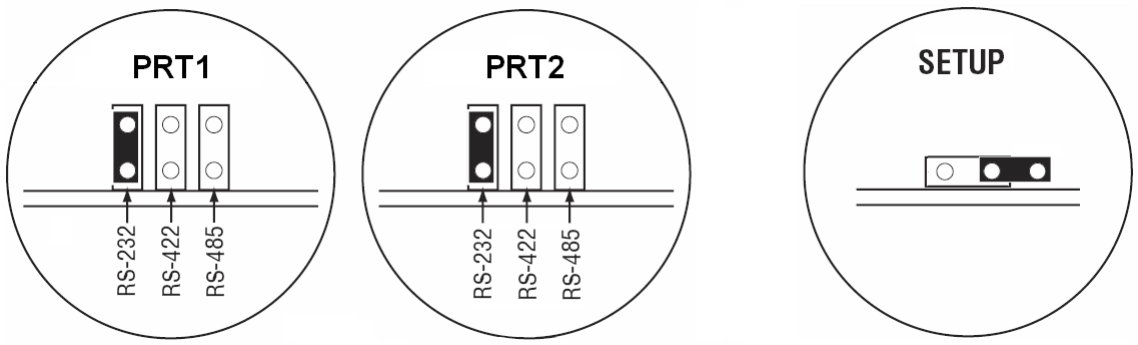
Qty.	Part Name	Part Number	Part Description
1	MVI56E-MCM Module	MVI56E-MCM	Modbus Communication Module
1	Cable	RL-CBL025	5 foot Ethernet Straight-Through Cable (Gray)
2	Cable	Cable #14, RJ45 to DB9 Male Adapter cable	For DB9 Connection to Module's Application Serial Port
2	Adapter	1454-9F	Two Adapters, DB9 Female to Screw Terminal. For RS422 or RS485 Connections to Port 1 and 2 of the Module
1	ProSoft Solutions DVD	DVD-001	Contains ProSoft Configuration Builder (PCB) and ProSoft Discovery Service (PDS) for the MVI56E-MCM modules.

If any of these components are missing, please contact ProSoft Technology Support for replacement parts.

1.4 Setting Jumpers

There are three jumpers located at the bottom of the module. The first two jumpers (P1 and P2) set the serial communication mode: RS-232, RS-422 or RS-485.

The following illustration shows the MVI56E-MCM jumper configuration, with the Setup Jumper OFF.



The Setup Jumper acts as "write protection" for the module's firmware. In "write protected" mode, the Setup pins are not connected, and the module's firmware cannot be overwritten. The module is shipped with the Setup jumper OFF. If you need to update the firmware, apply the Setup jumper to both pins.

Note: If you are installing the module in a remote rack, you may prefer to leave the Setup pins jumpered. That way, you can update the module's firmware without requiring physical access to the module.

1.5 Installing the Module in the Rack

Make sure your ControlLogix processor and power supply are installed and configured, before installing the MVI56E-MCM module. Refer to your Rockwell Automation product documentation for installation instructions.

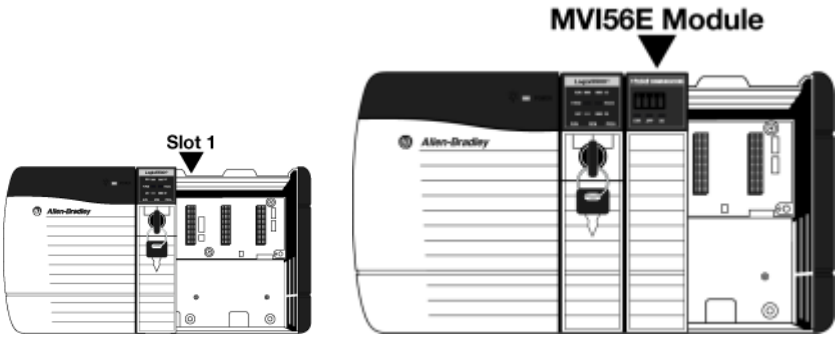
Warning: You must follow all safety instructions when installing this or any other electronic devices. Failure to follow safety procedures could result in damage to hardware or data, or even serious injury or death to personnel. Refer to the documentation for each device you plan to connect to verify that suitable safety procedures are in place before installing or servicing the device.

After you have checked the placement of the jumpers, insert the MVI56E-MCM into the ControlLogix chassis. Use the same technique recommended by Rockwell Automation to remove and install ControlLogix modules.

You can install or remove ControlLogix system components while chassis power is applied and the system is operating. However, please note the following warning.

Warning: When you insert or remove the module while backplane power is on, an electrical arc can occur. An electrical arc can cause personal injury or property damage by sending an erroneous signal to the system's actuators. This can cause unintended machine motion or loss of process control. Electrical arcs may also cause an explosion when they happen in a hazardous environment. Verify that power is removed or the area is non-hazardous before proceeding. Repeated electrical arcing causes excessive wear to contacts on both the module and its mating connector. Worn contacts may create electrical resistance that can affect module operation.

1. Align the module with the top and bottom guides, and then slide it into the rack until the module is firmly against the backplane connector.



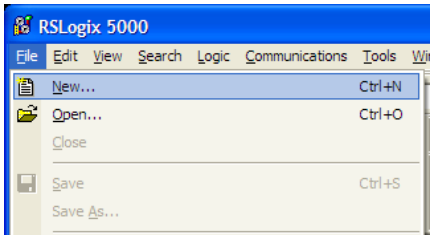
2 With a firm, steady push, snap the module into place.
3 Check that the holding clips on the top and bottom of the module are securely in the locking holes of the rack.
4 Make a note of the slot location. You must identify the slot in which the module is installed in order for the sample program to work correctly. Slot numbers are identified on the green circuit board (backplane) of the ControlLogix rack.
5 Turn power ON.

Note: If you insert the module improperly, the system may stop working or may behave unpredictably.

Note: When using the MVI56E-MCMXT, you must use the 1756-A5XT or 1756-A7LXT chassis. In these chassis, modules are spaced further apart than in standard ControlLogix chassis. Blank spacers are inserted between active modules.

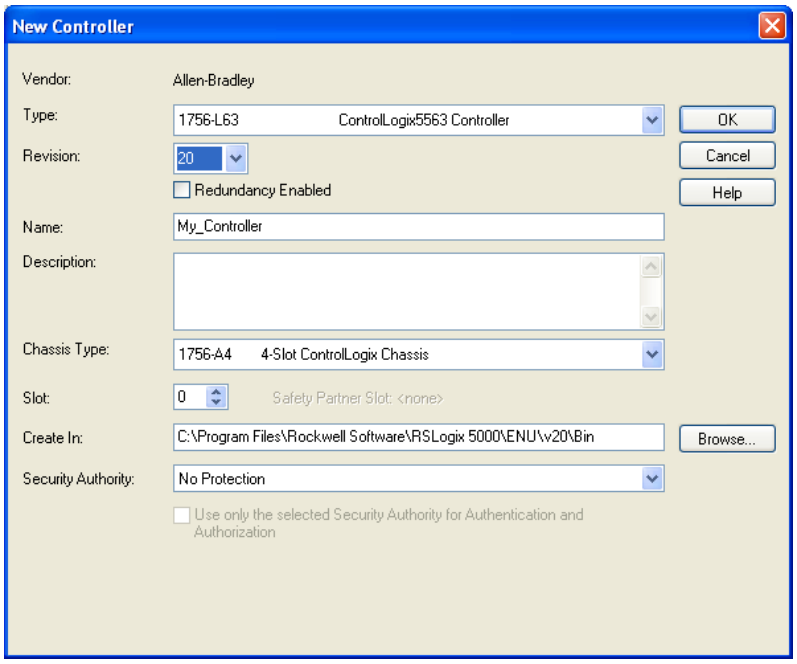
1.6 Creating a New RSLogix 5000 Project

1 Open the **FILE** menu, and then choose **NEW**.



2 Select your ControlLogix controller model.
3 Select the **REVISION** of your controller. Depending on the revision, there may be some small differences in the appearance of dialog boxes from the ones shown in this Guide.
4 Enter a name for your controller, such as *My_Controller*.
5 Select your ControlLogix chassis type.

6 Select **SLOT 0** for the controller.
7 Click **OK**



1.6.1 Before You Import the Add-On Instruction

Note: This section only applies if your processor is using RSLogix 5000 version 16 or higher. If you have an earlier version, please see Using the Sample Program - RSLogix 5000 Version 15 and earlier (page 180).

Two Add-On Instructions are provided for the MVI56E-MCM module. The first is required for setting up the module; the second is optional.

Copy the files from the *ProSoft Solutions DVD*, or download them from www.prosoft-technology.com. Save them to a convenient location in your PC, such as *Desktop* or *My Documents*.

File Name	Description
MVI56EMCM_AddOn_Rung_v2_8.L5X. A newer version may be available at www.prosoft-technology.com	L5X file containing Add-On Instruction, user defined data types, controller tags and ladder logic required to configure the MVI56E-MCM module
MVI56(E)MCM_Optional_AddOn_Rung_v1_2 .L5X. A newer version may be available at www.prosoft-technology.com	Optional L5X file containing additional Add-On Instruction with logic for changing Ethernet configuration and clock settings.

About the Optional Add-On Instruction

The Optional Add-On Instruction performs the following tasks:

- Read/Write Ethernet Configuration

  Allows the processor to read or write the module IP address, subnet mask, and network gateway IP address.

- Read/Write Module Clock Value

  Allows the processor to read and write the module clock settings. The module's free-running clock also stores the last time that the Ethernet configuration was changed or the last time the module was restarted or rebooted. The date and time of the last change or restart is displayed on the scrolling LED during module power-up/start-up sequence.

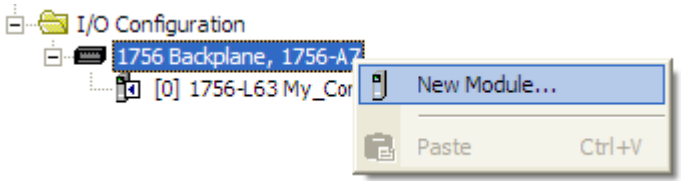
For more information, see Using the Optional Add-On Instruction (page 172).

Note: You can also set the date and time from the module's Connect to the Module's Web Page (page 113).
Important: The Optional Add-On Instruction supports only the two features listed above. You must use the regular MVI56E-MCM Add-On Instruction for all other features including backplane transfer and Modbus data communication.

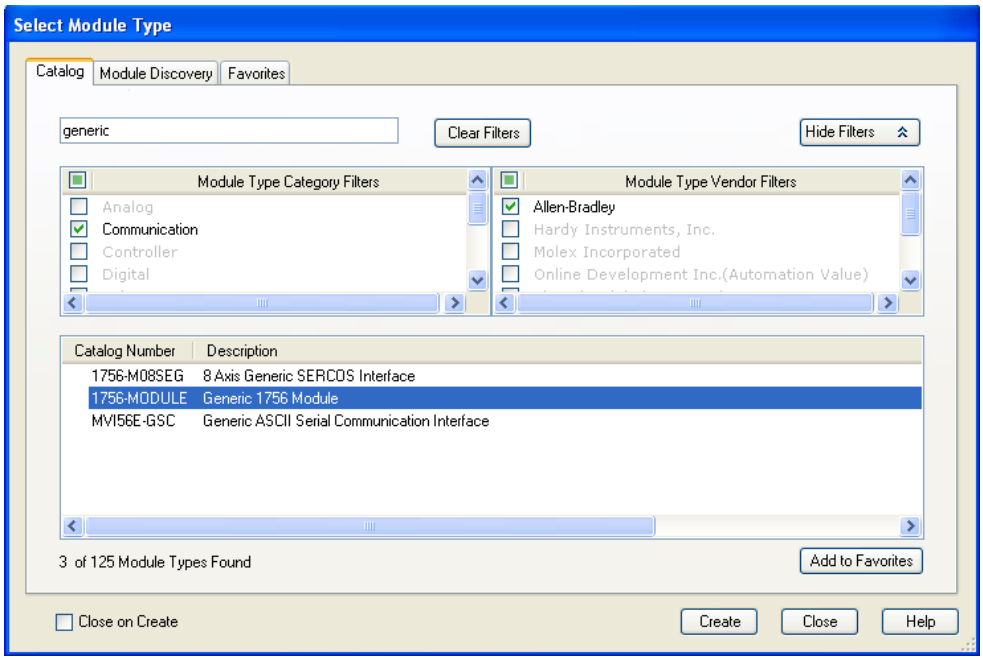
1.6.2 Creating the Module

1 Add the MVI56E-MCM module to the project.

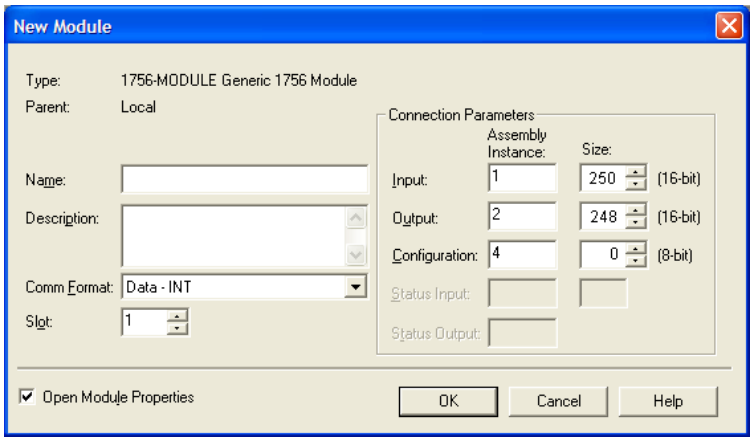
In the **CONTROLLER ORGANIZATION** window, select **I/O CONFIGURATION** and click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE...**



This action opens the **SELECT MODULE** dialog box. Enter *generic* in the text box and select the **GENERIC 1756 MODULE**. If you're using a controller revision of 15 or less, expand **OTHER** in the **SELECT MODULE** dialog box, and then select the **GENERIC 1756 MODULE**.



2 Click **CREATE.** This action opens the **NEW MODULE** dialog box.



3 In the **NEW MODULE** dialog box, enter the following values.

Parameter	Value
NAME	MCM
DESCRIPTION	Enter a description for the module. Example: Modbus Communication Module
COMM FORMAT	Select **DATA-INT**

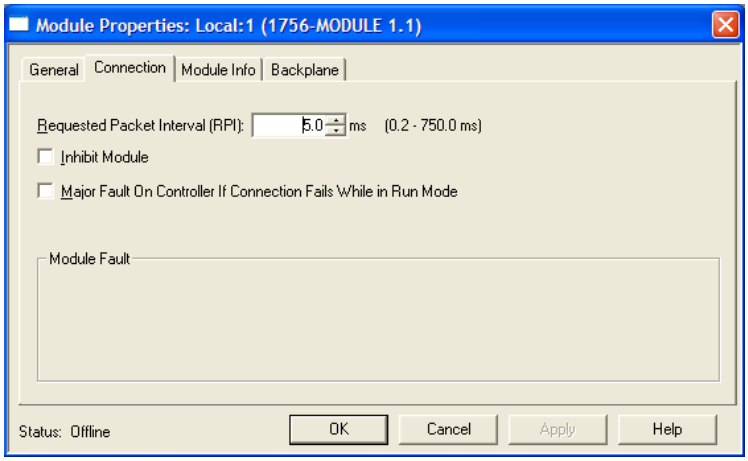
Parameter	Value
SLOT	Enter the slot number in the rack where the MVI56E-MCM module is located
INPUT ASSEMBLY INSTANCE	1
INPUT SIZE	250
OUTPUT ASSEMBLY INSTANCE	2
OUTPUT SIZE	248
CONFIGURATION ASSEMBLY INSTANCE	4
CONFIGURATION SIZE	0

Important: You must select the **COMM FORMAT** as **DATA - INT** in the dialog box, otherwise the module will not communicate over the backplane of the ControlLogix rack.

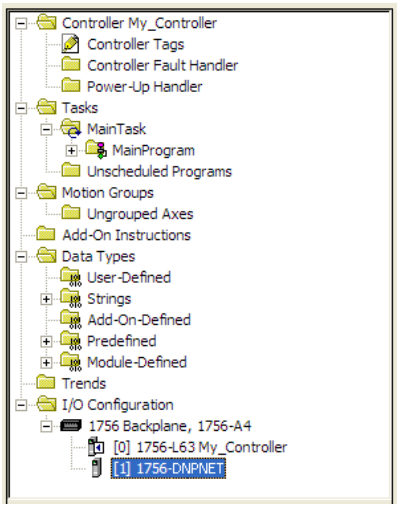
4 Click **OK** to continue.

5 Edit the Module Properties.

Select the **REQUESTED PACKET INTERVA**l value for scanning the I/O on the module. This value represents the minimum frequency at which the module will handle scheduled events. This value should not be set to less than 1 millisecond. The default value is 5 milliseconds. Values between 1 and 10 milliseconds should work with most applications.

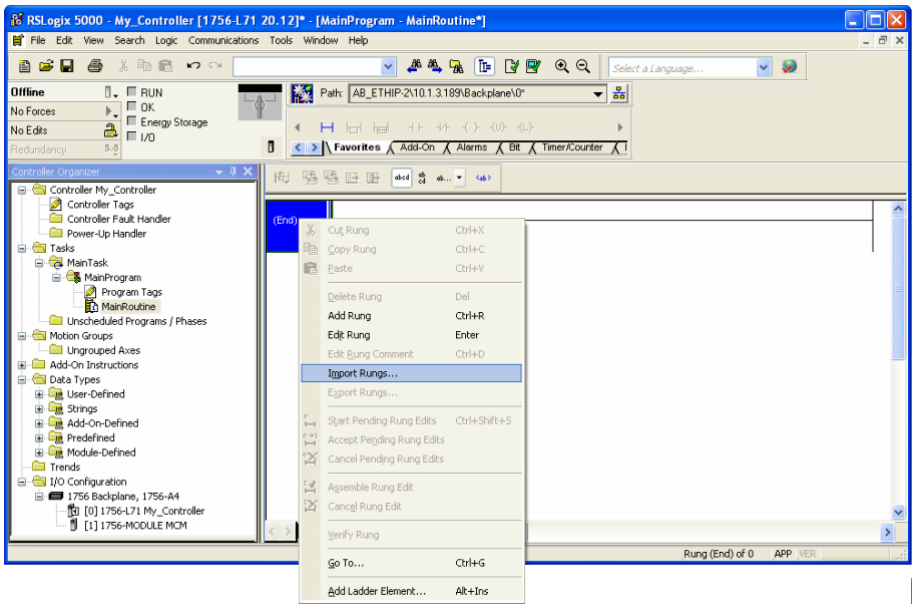


6 Click **OK** to save the module and close the dialog box. Notice that the module now appears in the **CONTROLLER ORGANIZATION** window.

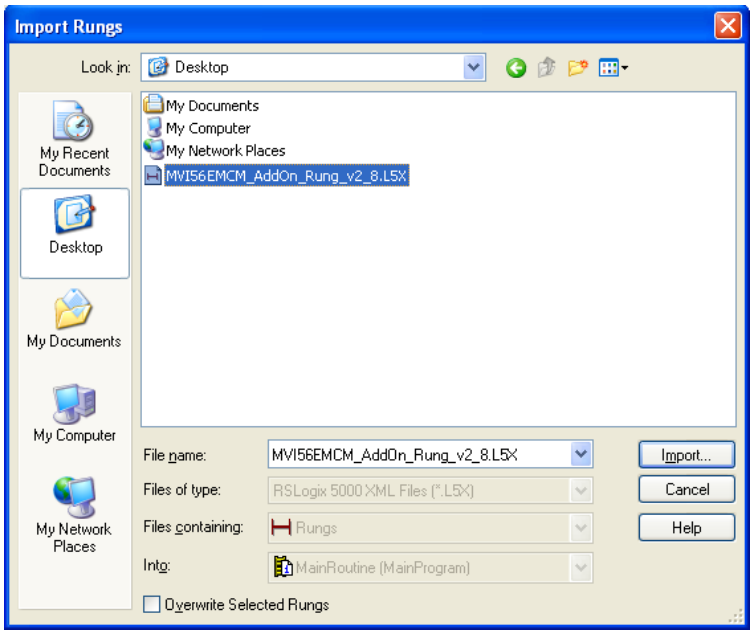


1.6.3 Import the Ladder Rung

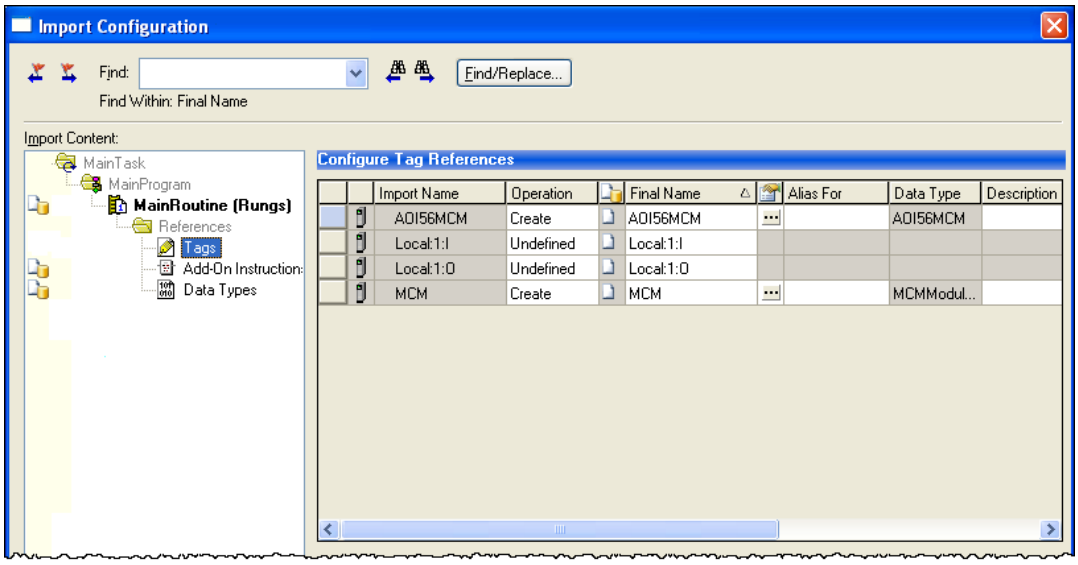
1 In the **CONTROLLER ORGANIZATION** window, expand the **TASKS** folder and subfolders until you reach the **MAINPROGRAM** folder.
2 In the **MAINPROGRAM** folder, double-click to open the **MAINROUTINE** ladder.
3 Select an empty rung in the routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNGS…**



4 Navigate to the location on your PC where you Before You Begin (page 15) the Add-On Instruction (for example, *My Documents* or *Desktop*). Select the **MVI56EMCM_ADDON_RUNG_V2.8.L5X** file.

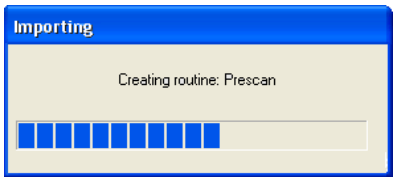


This action opens the **IMPORT CONFIGURATION** dialog box. Click **TAGS** under **MAINROUTINE** to show the controller tags that will be created. Note that if you are using a controller revision number of 16 or less, the **IMPORT CONFIGURATION** dialog box does not show the **IMPORT CONTENT** tree.

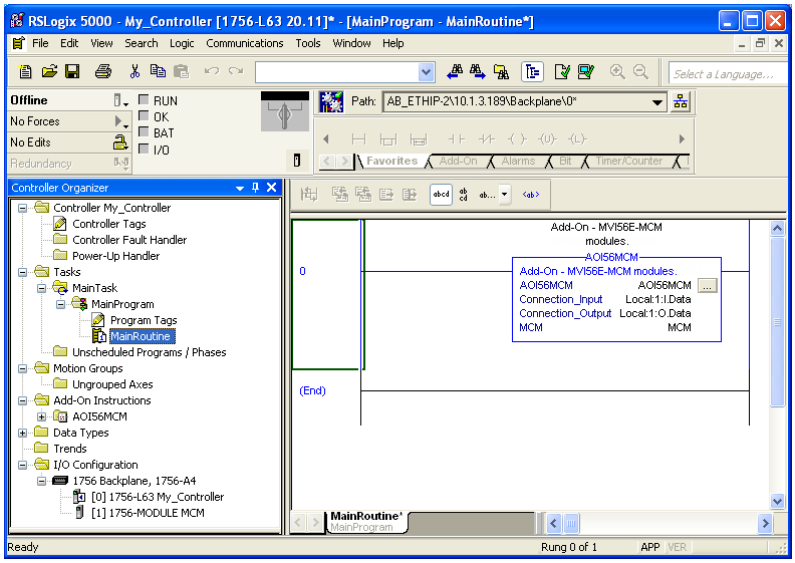


5 If you are using the module in a different slot (or remote rack), edit the connection input and output variables that define the path to the module. Edit the text in the **FINAL NAME** column **(NAME** column for controller revision 16 or less). For example, if your module is located in slot 3, change Local:1:I in the above picture to Local:3:I. Do the same for Local:1:O. If your module is located in Slot 1 of the local rack, this step is not required.

6 Click **OK** to confirm the import. RSLogix will indicate that the import is in progress:



When the import is completed, the new rung with the Add-On Instruction will be visible as shown in the following illustration.



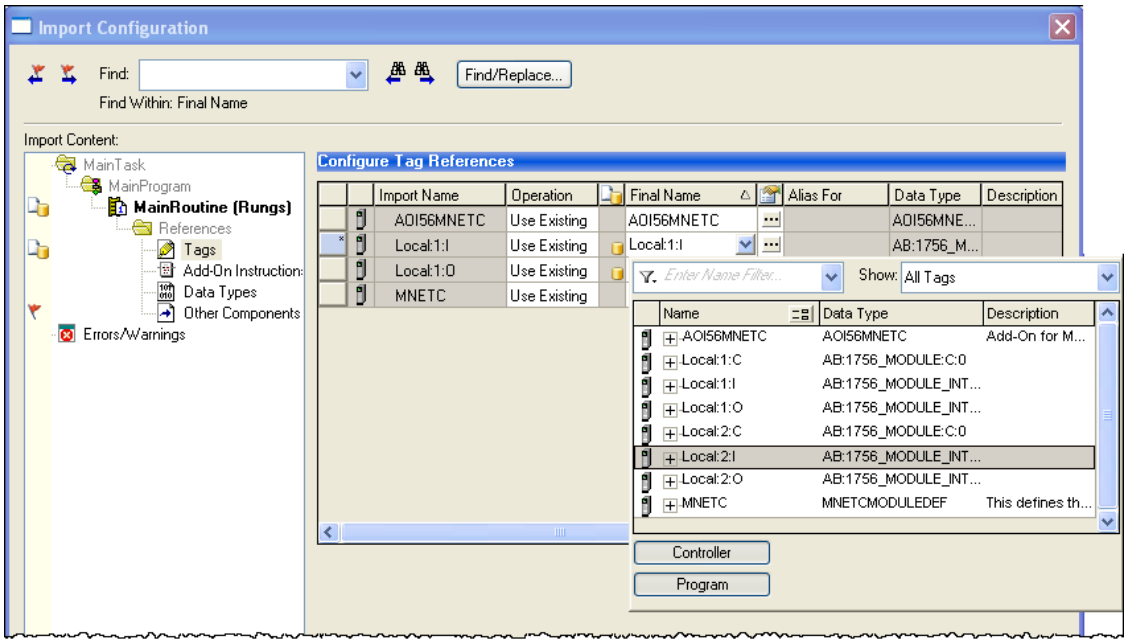
The procedure has also imported new User Defined Data Types, Controller Tags, and the Add-On Instruction for your project.

7 Save the application and then download the sample ladder logic into the processor.

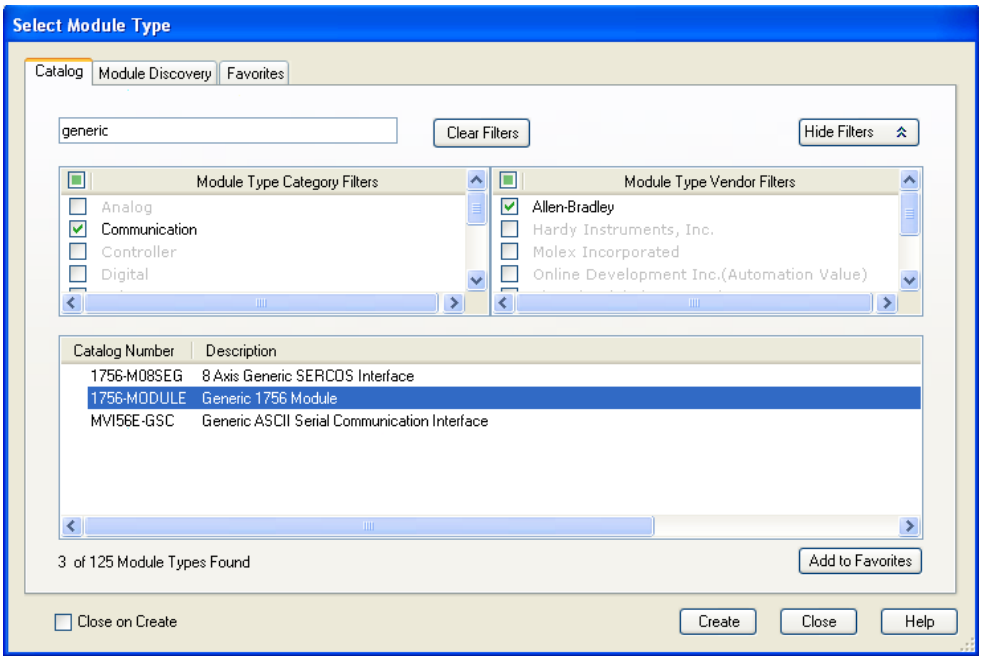
1.6.4 Adding Multiple Modules (Optional)

Important: If your application requires more than one MVI56E-MCM module in the same project, follow the steps below.

1 In the **I/O CONFIGURATION** folder, click the right mouse button to open a shortcut menu, and then choose **NEW MODULE**.



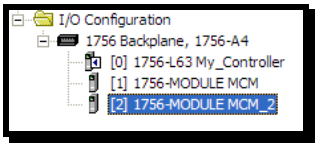
2 Select **1756-MODULE**. If you're using a controller revision of 16 or less, expand **OTHER** in the **SELECT MODULE** dialog box, and then select the **1756-MODULE**.



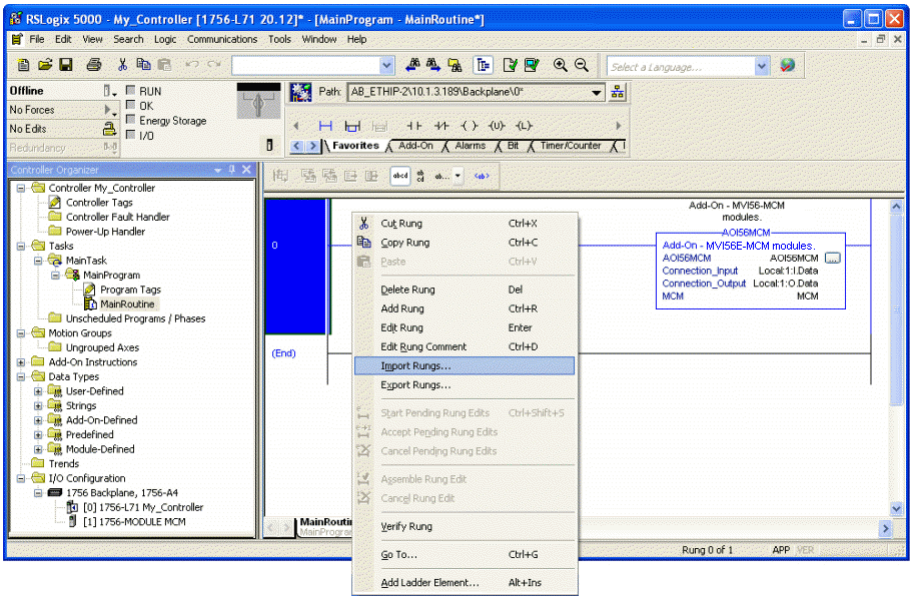
3 Fill the module properties as follows:

Parameter	Value
NAME	Enter a module identification string. Example: MCM_2.
DESCRIPTION	Enter a description for the module. Example: ProSoft Modbus Communication Module.
COMM FORMAT	Select **DATA-INT.**
SLOT	Enter the slot number in the rack where the MVI56E-MCM module is located.
INPUT ASSEMBLY INSTANCE	1
INPUT SIZE	250
OUTPUT ASSEMBLY INSTANCE	2
OUTPUT SIZE	248
CONFIGURATION ASSEMBLY INSTANCE	4
CONFIGURATION SIZE	0

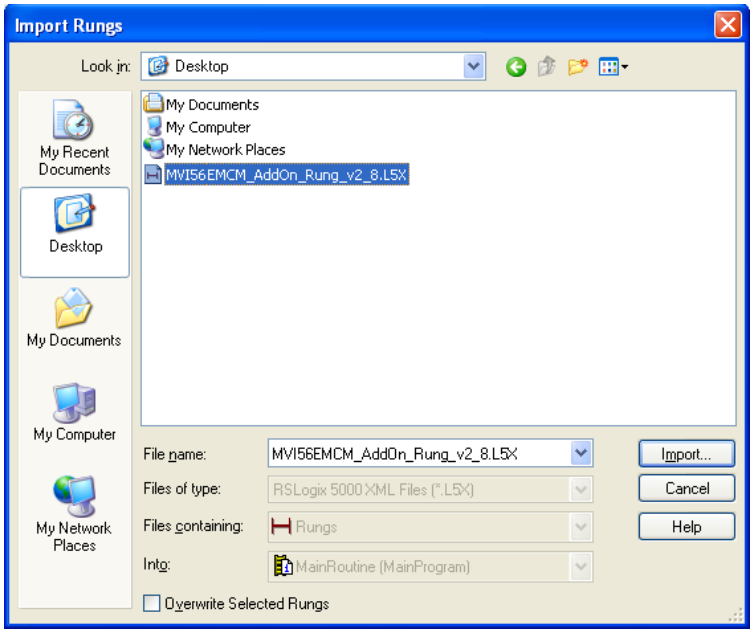
4 Click **OK** to confirm. The new module is now visible:



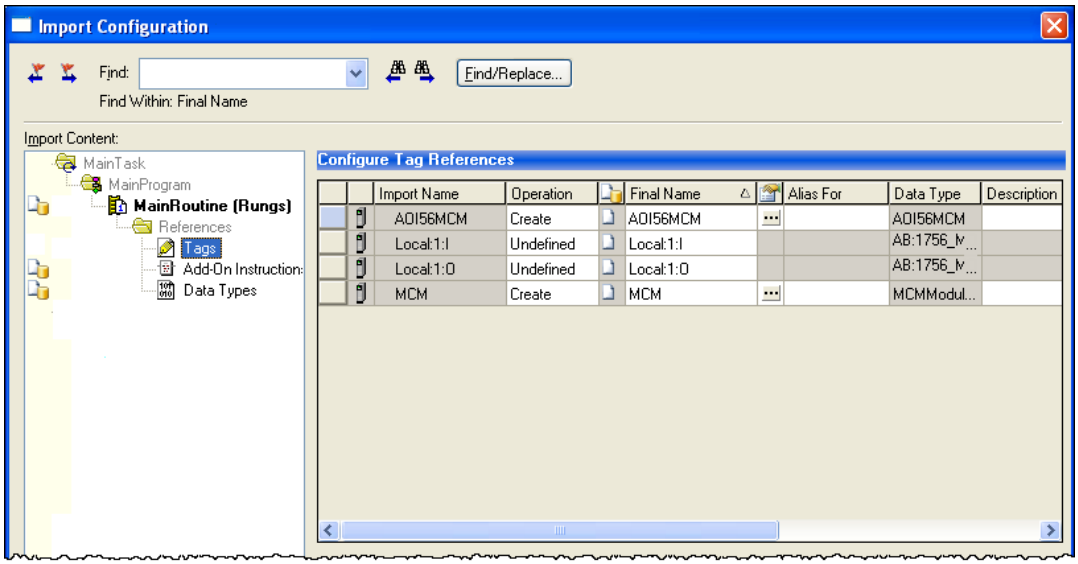
5 Expand the **TASKS** folder, and then expand the **MAINTASK** folder.
6 In the **MAINPROGRAM** folder, double-click to open the **MAINROUTINE** ladder.
7 Select an empty rung in the routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNGS…**



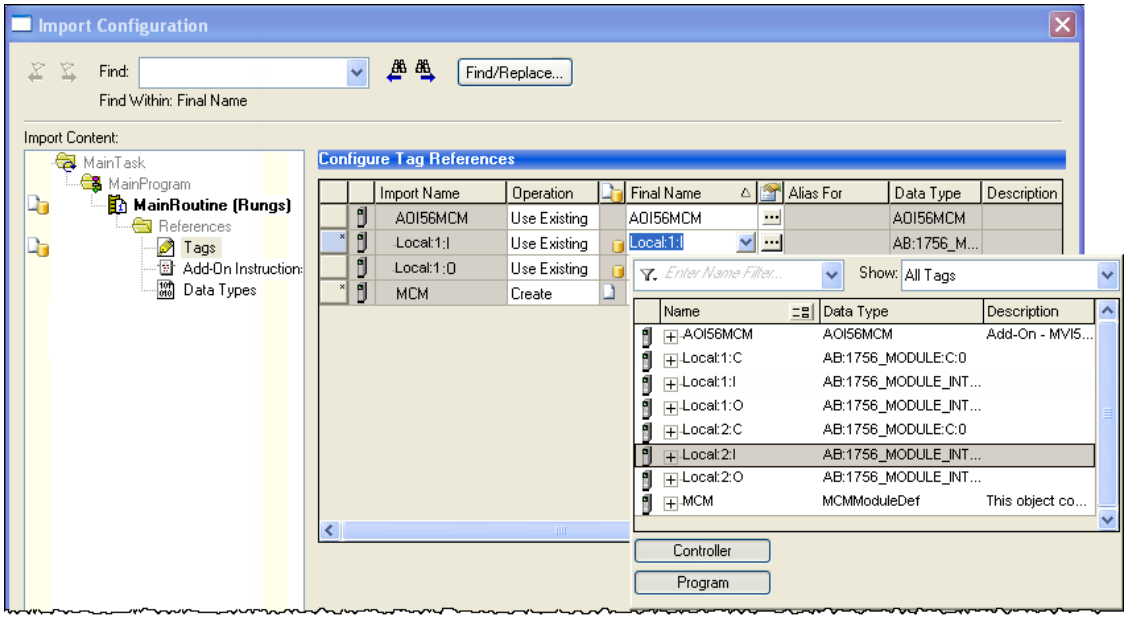
8 Select the **MVI56EMCM_ADDON_RUNG_V2_8.L5X** file, and then click **IMPORT.**



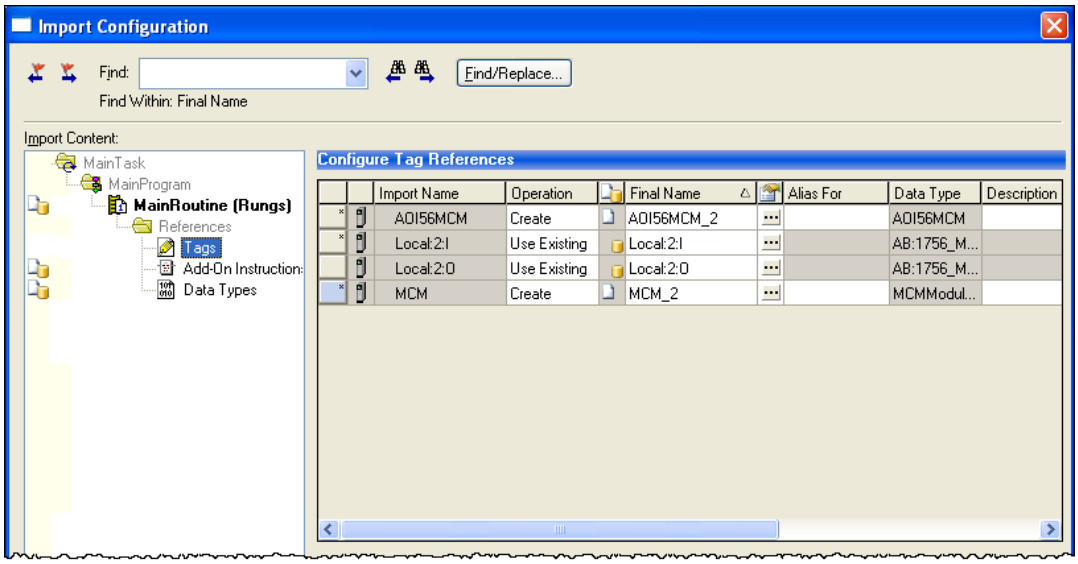
9 This action opens the **IMPORT CONFIGURATION** window. Click **TAGS** under **MAINROUTINE** to show the tags that will be imported.



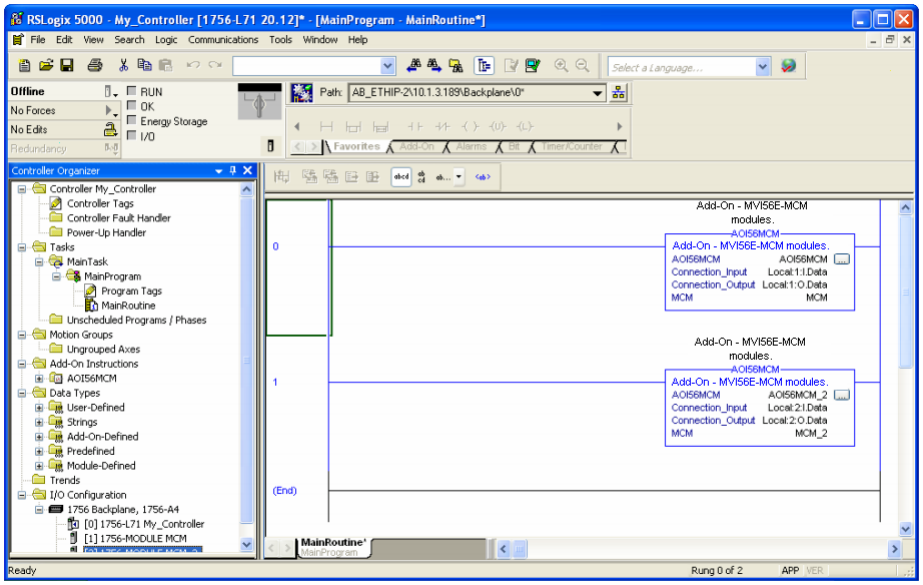
10 Associate the I/O connection variables to the correct module. The default values are Local:1:I and Local:1:O so you may have to edit the **FINAL NAME** field to change the values. You can also click the drop-down arrow to select the correct name.



11 Change the default tags **MCM** and **AOI56MCM** to avoid conflict with existing tags. In this step, append the string "_2", as shown in the following illustration.



12 Click **OK** to confirm.



The setup procedure is now complete. Save the project and download the application to your ControlLogix processor.

1.6.5 Adjust the Input and Output Array Sizes (Optional)

The module internal database is divided into two user-configurable areas:

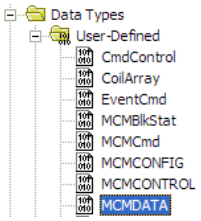
- Read Data
- Write Data.

The Read Data area is moved from the module to the processor, while the Write Data area is moved from the processor to the module. You can configure the start register and size of each area. The size of each area you configure must match the Add-On Instruction controller tag array sizes for the **READDATA** and **WRITEDATA** arrays.

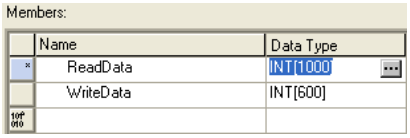
The MVI56E-MCM sample program is configured for 600 registers of **READDATA** and 600 registers of **WRITEDATA,** which is sufficient for most application. This topic describes how to configure user data for applications requiring more than 600 registers of ReadData and WriteData. In this example, we will expand both the Read and Write Data sizes to 1000.

Important: Because the module pages data in blocks of 200 registers at a time, you must configure your user data in multiples of 200 registers.
Caution: When you change the array size, RSLogix may reset the MCM tag values to zero. To avoid data loss, be sure to save your settings before continuing.

1 In the **CONTROLLER ORGANIZATION** window, expand the **DATA TYPES** and **USER-DEFINED** folders, and then double-click **MCMDATA.** This action opens an edit window for the MCMDATA data type.



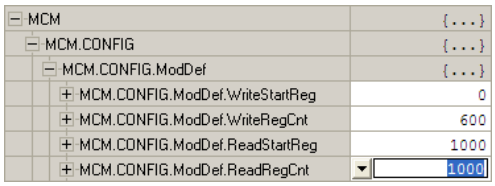
2 In the edit window, change the value of the **READDATA** array from **INT[600]** to **INT[1000]** as shown, and then click **APPLY.**



Note: If RSLogix resets your data values, refer to the backup copy of your program to re-enter your configuration parameters.

3 Next, navigate to **CONTROLLER TAGS** and double click to open an edit window. Click the **MONITOR TAGS** tab at the bottom of the edit window.

4 Click **[+]** to expand the **MCM.CONFIG.MODDEF** section, and then change the **READREGCNT** parameter from 600 to 1000.



5 Save and download the sample program to the processor.
6 Go Online with the ControlLogix processor, and then toggle the **MCM.CONTROL.WARMBOOT** bit to download the configuration to the MVI56E-MCM module.

Note: Any changes made to the MCM.CONFIG or WriteData arrays must be downloaded to the MVI56E-MCM module. The use of the MCM.CONTROL.WarmBoot or MCM.CONTROL.ColdBoot bit will cause the MVI56E-MCM module to re- read the configuration from the ControlLogix processor.

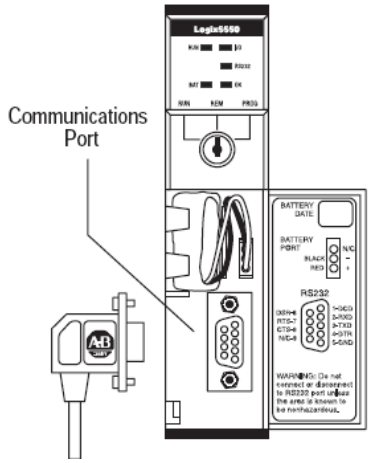
To modify the **WRITEDATA** array, follow the steps in this topic, but substitute **WRITEDATA** for ReadData throughout. Also, make sure that the **READDATA** and **WRITEDATA** arrays do not overlap in the module memory. For example, if your application requires 2000 words of WriteData starting at register 0, then your **MCM.CONFIG.MODDEF.READSTARTREG** must be set to a value of 2000 or greater.

1.7 Connecting Your PC to the ControlLogix Processor

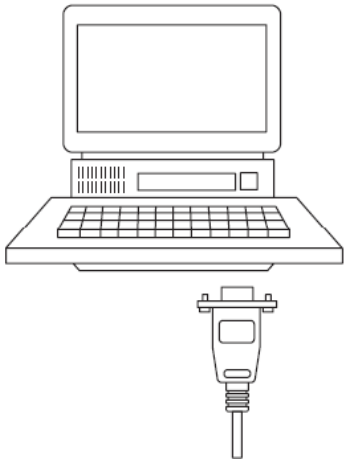
There are several ways to establish communication between your PC and the ControlLogix processor. The following steps show how to establish communication through the serial interface.

Note: It is not mandatory that you use the processor's serial interface. You may access the processor through whatever network interface is available on your system. Refer to your Rockwell Automation documentation for information on other connection methods

1 Connect the right-angle connector end of the cable to your controller at the communications port.



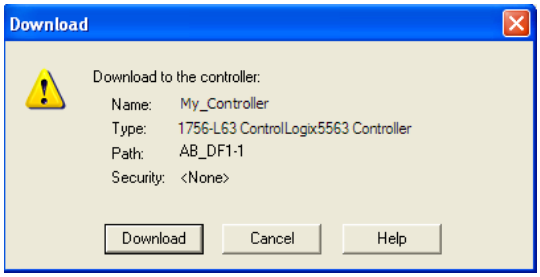
2 Connect the straight connector end of the cable to the serial port on your computer.



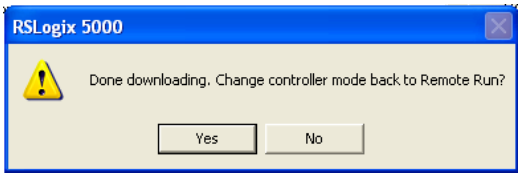
1.8 Downloading the Sample Program to the Processor

Note: The key switch on the front of the ControlLogix processor must be in the REM or PROG position.

1. If you are not already online with the processor, in RSLogix 5000 open the *Communications* menu, and then choose **DOWNLOAD**. RSLogix 5000 will establish communication with the processor. You do not have to download through the processor's serial port, as shown here. You may download through any available network connection.
2. When communication is established, RSLogix 5000 will open a confirmation dialog box. Click the **DOWNLOAD** button to transfer the sample program to the processor.



3. RSLogix 5000 will compile the program and transfer it to the processor. This process may take a few minutes.
4. When the download is complete, RSLogix 5000 will open another confirmation dialog box. If the key switch is in the REM position, click **OK** to switch the processor from PROGRAM mode to RUN mode.

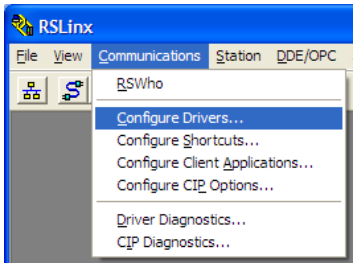


Note: If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

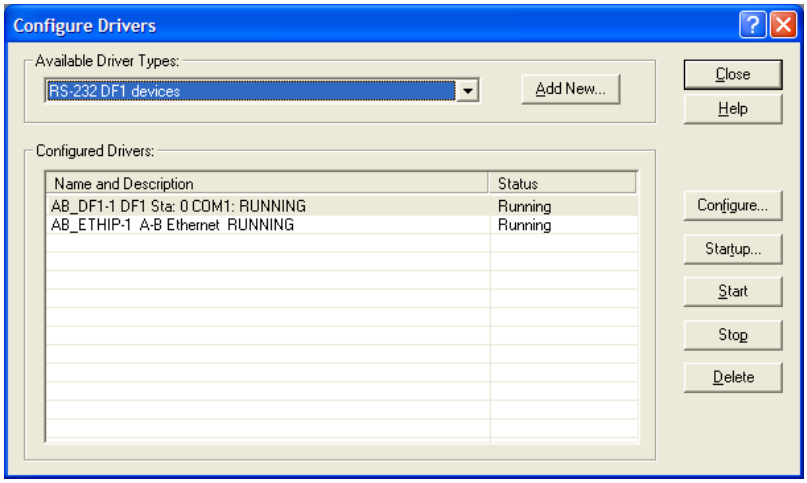
1.8.1 Configuring the RSLinx Driver for the PC COM Port

When trying to connect serially, if RSLogix is unable to establish communication with the processor, follow these steps.

1 Open *RSLinx*.
2 Open the **COMMUNICATIONS** menu, and click **CONFIGURE DRIVERS**.

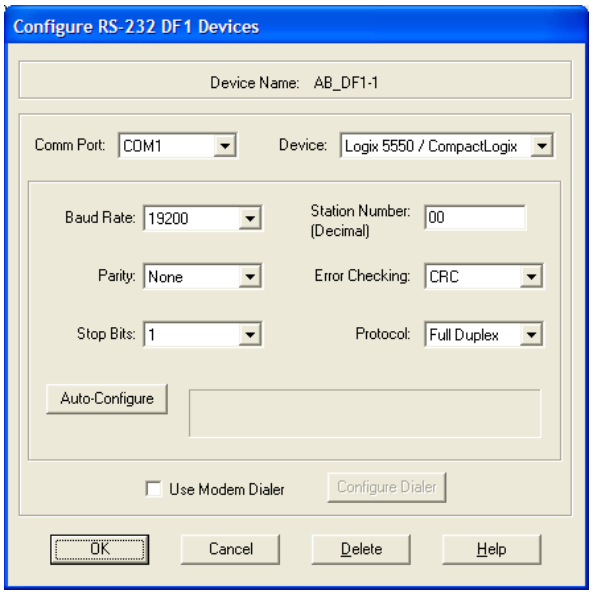


This action opens the *Configure Drivers* dialog box.



Note: If the list of configured drivers is blank, you must first choose and configure a driver from the *Available Driver Types* list. The recommended driver type to choose for serial communication with the processor is *RS-232 DF1 Devices*.

3 Click to select the driver, and then click **CONFIGURE**. This action opens the *Configure RS-232 DF1 Devices* dialog box.



4 Click the **AUTO-CONFIGURE** button. RSLinx will attempt to configure your serial port to work with the selected driver.
5 When you see the message *Auto Configuration Successful*, click the **OK** button to dismiss the dialog box.

Note: If the auto-configuration procedure fails, verify that the cables are connected correctly between the processor and the serial port on your computer, and then try again. If you are still unable to auto-configure the port, refer to your RSLinx documentation for further troubleshooting steps.

2 Configuration as a Modbus Master

In This Chapter

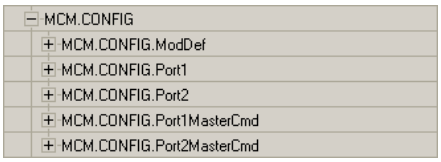
- Overview 33
- ModDef Settings 34
- Master Command Examples 41
- Floating-Point Data Handling (Modbus Master) 46
- Command Control and Event Command 52

2.1 Overview

This section describes how to configure the module as a **MODBUS MASTER** device. The Master is the only device on a Modbus network that can initiate communications. A Master device issues a request message, and then waits for the Slave to respond. When the Slave responds, or when a timeout has occurred, the Modbus Master will then execute the next command in the list.

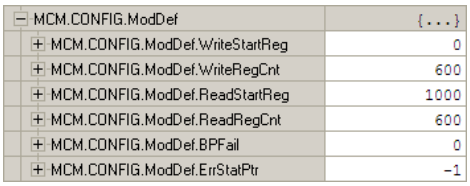
The following RSLogix controller tags contain the Modbus Master configuration. You must configure all three sets of controller tags.

1. The **MODDEF** controller tags set up the backplane communication between the MVI56E-MCM module and the ControlLogix processor. These settings include register addresses for ReadData and WriteData. You can configure up to 10,000 data registers in the module to exchange data with the ControlLogix processor.
2. The **PORT1** and **PORT 2** controller tags configure the Modbus application serial port. This set of controller tags configures serial communication parameters such as baud rate, data bits, and stop bits. They also contain settings to configure the port as a Modbus Master or a Modbus Slave.
3. The **PORT1MASTERCOMMAND** and **PORT2MASTERCOMMAND** controller tags define a polling table (command list) for the Modbus Master. This set of tags contains the addresses for devices on the network, the types of data (Modbus Function Codes) to read and write with those devices, and the location to store the data within the module's 10,000 data registers.



2.2 ModDef Settings

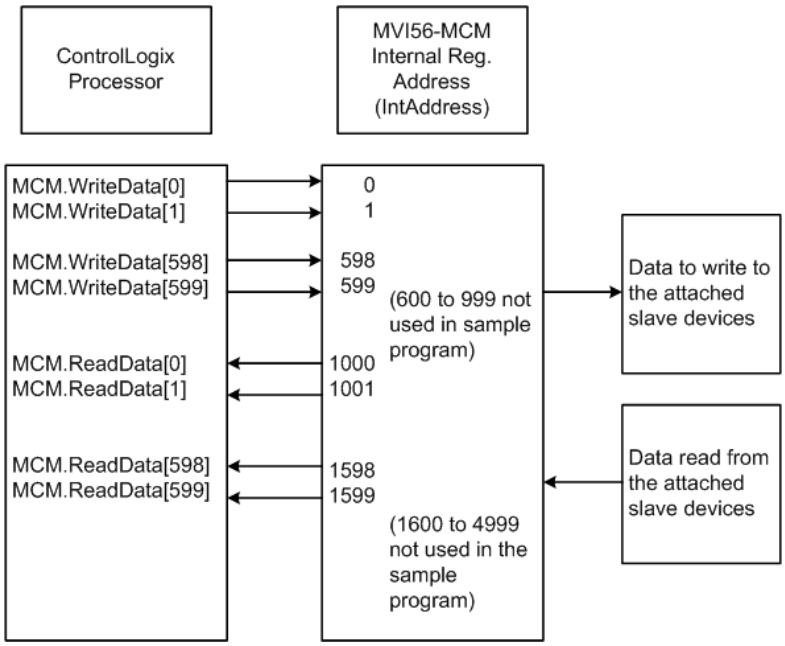
The **MCM.CONFIG.MODDEF** tag defines the 10,000 data registers to use for read and write data within the MVI56E-MCM module. You will use these data read and write locations in the IntAddress tag within each Master Command Configuration (page 37). The following illustration shows the values from the sample program.



The **WRITESTARTREG** tag determines the starting register location for the **WRITEDATA[0** to **599]** array. The **WRITEREGCNT** tag determines how many of the 10,000 registers to use to send data to the module. The sample ladder file uses 600 registers for write data, labeled **MCM.WRITEDATA[0** to **599].**

Label	Description
WRITESTARTREG	Specifies where in the 10,000 register module memory to place data sent from the WriteData tags in the ControlLogix processor.
WRITEREGCNT	Specifies how many registers of data the MVI56E-MCM module will request from the ControlLogix processor. Because the module pages data in blocks of 200 words, this number must be evenly divisible by 200.
READSTARTREG	Specifies which registers in the module's read data area to send to the ReadData tags in the ControlLogix processor.
READREGCNT	Sets how many registers of data the MVI56E-MCM module will send to the ControlLogix processor. This value should also be a multiple of 200.
BPFAIL	Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network. Typically used when the module is configured as a Slave.
ERRSTATPTR	Also used mainly when the module is configured as a Slave. This parameter places the STATUS data into the database of the module.

The sample configuration values configure the module database to store **WRITEDATA[0** to **599]** in registers 0 to 599, and **READDATA[0 TO 599]** in registers 1000 to 1599, as shown in the following illustration.



2.2.1 Port Configuration

The **MCM.CONFIG.PORTX** controller tags are used when the module is configured as a Modbus Master device. Port 1 and Port 2 each have their own set of parameters to configure.

Note: Any changes made within the **MCM.CONFIG** array must be downloaded to the MVI56E-MCM module by setting the **WARMBOOT** or **COLDBOOT** bit, or cycling power to the module.

Any parameters not mentioned in this section are not used when the module is configured as a Modbus Master.

Verify that you are in **MONITOR TAGS** mode. Then use the scroll bar at the bottom of the window to view a description of each parameter. The following table uses that information.

Parameter	Description
Enabled	**1 = ENABLE PORT**, **0 = DISABLE PORT**
Type	**0=MASTER**, **1=SLAVE**, **2=SLAVE WITH UNFORMATTED PASS-THROUGH**, **3=SLAVE, FORMATTED PASS-THROUGH WITH DATA SWAPPING**, **4= SLAVE, FORMATTED PASS-THROUGH WITH NO DATA SWAPPING.**
Protocol	**0 = MODBUS RTU MODE**, **1 = MODBUS ASCII MODE**
Baudrate	Sets the baud rate for the port. Valid values for this field are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 384 or 3840 (for 38,400 baud), 576 or 5760 (for 57,600 baud) and 115,1152, or 11520 (for 115,200 baud)
Parity	0 = None, 1 = Odd, 2 = Even
DataBits	Modbus RTU mode = 8 Modbus ASCII mode = 8 or 7
StopBits	Valid values are 1 or 2.
RTS On	0 to 65535 milliseconds to delay after RTS line is asserted on the port before data message transmission begins. This delay can be used to allow for radio keying or modem dialing before data transmission begins.
RTS Off	0 to 65535 milliseconds to delay after data message is complete before RTS line is dropped on the port.
Use CTS Line	No or Yes This parameter is used to enable or disable hardware handshaking. The default setting is No hardware handshaking, CTS Line not used. Set to No if the connected devices do not need hardware handshaking. Set to Yes if the device(s) connected to the port require hardware handshaking (most modern devices do not) If you set this parameter to Yes, be sure to pay attention to the pinout and wiring requirements to be sure the hardware handshaking signal lines are properly connected; otherwise communication will fail.
CmdCount	0-325 commands This parameter sets the number of commands to execute from the command list. Setting to zero (0) will disable all command polling. Setting to a value less than the number of configured commands will limit polling to the number of commands specified by this parameter. Setting to a value greater than the number of configured commands will cause invalid command errors to be reported for the unconfigured commands.
Minimum Command Delay	0-65535 milliseconds The amount of delay in milliseconds to be inserted after receiving a Slave response or encountering a response timeout before retrying the command or sending the next command on the list. Use this parameter to slow down overall polling speed and spread out commands on networks with Slaves that require additional gaps between messages.
CmdErrPtr	Internal DB location to place command error list Each command will reserve one word for the command error code for that command. See Verify Communication (page 66). **CMDERRPTR** value should be within the range of the **READDATA** array. See Backplane Configuration (page 103).

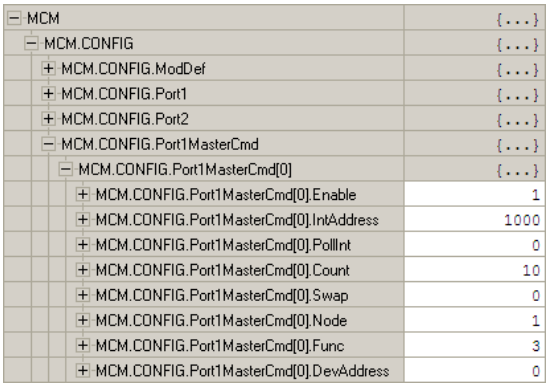
Parameter	Description
Error Delay Counter	This parameter specifies the number of poll attempts to be skipped before trying to re-establish communications with a slave that has failed to respond to a command within the time limit set by the *Response Timeout* parameter. After the slave fails to respond, the master will skip sending commands that should have been sent to the slave until the number of skipped commands matches the value entered in this parameter. This creates a sort of *slow poll* mode for slaves that are experiencing communication problems.
RespTO	0 to 65535 milliseconds response timeout for command before it will either reissue the command, if **RETRYCOUNT > 0.** If the RetryCount =0 or if the designated number of retries have been accomplished, then the Master will move on to the next command in the list.
RetryCount	Number of times to retry a failed command request before moving to the next command on the list.

Note: To use up to 325 commands, your MVI56E-MCM module needs to have firmware version 3.01 or higher, and your MVI56E-MCM Add-On Instruction needs to be version 2.8 or higher. Earlier versions support up to 100 commands.

2.2.2 Master Command Configuration

This topic describes the communications with the Master Port and the Slave devices that are connected to that port.

Verify you are in **MONITOR TAGS** mode. Then use the scroll bar at the bottom of the window to view a description of each parameter.



Label	Description
Enable	0 = Disabled Command will not be executed, but can be enabled using command control option in ladder logic. 1 = Enabled Command is enabled and will be sent out to the target device. 2 = Conditional Write Only for Func 5, 15, 6, or 16 data will be sent to the target device only when the data to be written has changed.
IntAddress	Determines where in the module's 10,000-register database the data will be stored to or written from. For a Read command, this will determine, after information has been read from a Slave, where it will be placed in the module database. For read commands, you should configure this for a location that is configured for **READDATA.** The internal database location for the **READDATA** and **WRITEDATA** tags is determined by the configuration in the MCM.ModDef tag location. For write data, the **INTADDRESS** determines where to obtain the information to write to the Slave device. This must be a location that is configured as **WRITEDATA.** **Note:** When using a bit level command, you must define this field at the bit level. For example, when using a function code 1, 2 for a Read command, you must have a value of 16000 to place the data in MCM.ReadData[0] (register 1000 * 16 bits per register = 16000). This controller tag is a 16bit signed integer. This means you can only enter values of -32768 to 32767 in the tag. If a value to be entered is above the 32767 (but below 65535) threshold, it will display as a negative value in the tag. Simply subtract 65536 from the value to get the 'acceptable' value to enter into the tag. **Example:** You need to use an Internal bit Address of 48000, but you cannot enter '48000' into the tag because it causes an error. 48000 - 65536 = -17536 You need to enter -17536 in the Internal Address parameter for this command.
PollInt	The Poll Interval (PollInt) is the number of seconds that a Master device will wait before issuing this command.
Count	Sets how many continuous words (Function Codes 3, 4, and 16) or bits (Function Codes 1, 2, and 15) to request from the Slave device. Valid values are 1 to 125 words for function codes 3, 4, and 16, while you can specify a range of 1 to 2000 for function codes 1, 2, and 15. **Note:** These values are the maximum allowed in the Modbus protocol. Some devices may support fewer words or bits than the maximum allowed.
Swap	Typically used when reading floating-point data, swaps the data read from the Slave device before it is placed into the module memory. For example, you receive 4 bytes of data from the Slave (ABCD). 0 = No swapping (ABCD) 1 = Word pairs switched (CDAB) 2 = Bytes and words switched (DCBA) 3 = Bytes swapped (BADC)
Node	Node address of the device on the network to read data from, or write data to. Valid addresses are 1 to 247. Address 0 is reserved for broadcast write commands (will broadcast a Write command to all devices on the network).

Label	Description
Func	Determines the Modbus function code that to issue in the command to the Slave device. Valid values for this field are as follows: **1 = Read Coil Status** This will read Modbus addresses 0001 to 9999. These bit values indicate coil status. Use Function Code 5 or 15 to write to these registers **2 = Read Input Coils** This will read Modbus addresses 10001 to 29999. Like Function Code 1, these are also bit values, but Function Code 2 values are Read Only data values. Use Function Code 5 or 15 to write to these registers **3 = Read Holding Registers** This will read Modbus addresses 40001 to 47999. This is a 16-bit word value. Use Function Codes of 6 and 16 to write to these registers. **4 = Read Input Registers** This will read Modbus addresses 30001 to 39999. These are also 16-bit word values, but are Read Only data. The Modbus Master cannot write to these registers. **5 = Force (Write) Single Coil Status** This will write to Modbus addresses 0001 to 9999. This command will write to only one coil. Use Function Code 15 to write to multiple coils. **6 = Preset (Write) Single Register** This will write to Modbus addresses 40001 to 47999. This command writes a single register value out to a Slave device. Use Function Code 16 to write to multiple registers. **15 = Force (Write) Multiple Coil** This will write multiple coil values to the Slave addresses 0001 to 9999. **16 = Preset (Write) Multiple Register** This will write multiple register values to the Slave device at addresses 40001 to 49999.

Label	Description
DevAddress	Specifies the Modbus Slave address for the registers associated with that command. This is the offset address for the Modbus Slave device. With Modbus, to read an address of 40001, what will actually be transmitted out port is Function Code 03 (one byte) with an address of 00 00 (two bytes). This means that to read an address of 40501, use Func 3 with a DevAddress of 500. This applies to Modbus addresses 10001 to 47999. Below is a definition that will help with your DevAddress configuration: **Function Codes 1, 5, or 15** ▪ DevAddress = Modbus address in device - 0001 Example: Modbus address 0001 = DevAddress 0 ▪ Modbus address 1378 = DevAddress 1377 **Function Code 2** ▪ DevAddress = Modbus address in device - 10001 Example: Modbus address 10001 = DevAddress 0 ▪ Modbus address 10345 = DevAddress 344 **Function Codes 3, 6, or 16** ▪ DevAddress = Modbus address in device - 40001 Example: Modbus address 40001 = DevAddress 0 ▪ Modbus address 40591 = DevAddress 590 **Function Code 4** ▪ DevAddress = Modbus address in device - 30001 Example: Modbus address 30001 = DevAddress 0 ▪ Modbus address 34290 = DevAddress 4289

2.2.3 Other Modbus Addressing Schemes

While the above information will handle most devices, some device manufacturers show their Modbus addressing differently.

The two most common schemes are six-digit addressing (400101, 301000, etc…) and some devices show their addressing already as an offset address (the address that actually goes out on the Modbus communication line). This is an example.

Actual Values (Input Registers) Addresses: 0200 to 0E1F

STATUS	0200	Switch Input Status
	0201	LED Status Flags
	0202	LED Attribute Flags
	0203	Output Relay Status Flags

If your device manufacturer gives you addressing like this, "Input Registers" then you will use Function Code 4, and then place the address shown in the DevAddress field. Also, most manufacturers that show this type of addressing will list the address in hex, as is the case with the device shown above. So for this example device, use Func = 4 (Input Registers) with a DevAddress of 512 decimal (200h) to read the "Switch Input Status" value.

Why does my Slave show addressing such as 400,001 or 301,345?

For the 6 digit addressing, use the same function codes and configuration as configured above, just the starting address has changed.

Below is a definition that will help with your DevAddress configuration:

Function Codes 1, 5, or 15 **DevAddress** = Modbus address in device - 0001

- Example: Modbus address 0001 = DevAddress 0
- Modbus address 1378 = DevAddress 1377

Function Code 2 **DevAddress =** Modbus address in device - 100001

- Example: Modbus address 100001 = DevAddress 0
- Modbus address 100345 = DevAddress 344

Function Codes 3, 6, or 16 **DevAddress** = Modbus address in device - 400001

- Example: Modbus address 400001 = DevAddress 0
- Modbus address 400591 = DevAddress 590

Function Code 4 **DevAddress** = Modbus address in device - 300001

- Example: Modbus address 300001 = DevAddress 0
- Modbus address 304290 = DevAddress 4289

For example, our device listed above could show their addressing as follows.

Variable Name	Data Type	Address
Switch_Input_Status	INT	300513
LED_Status_Flags	INT	300514
LED_Attribute_Flags	INT	300515
Output_Relay_Status_Flags	INT	300516

To read the same parameter "Switch_Input_Status", you would still issue a Function Code 4, and use a DevAddress of 512 decimal.

2.3 Master Command Examples

2.3.1 Read Holding Registers 4x (Modbus Function Code 3)

The 4x holding registers are used for Analog Values such as Pressure, Temperature, Current, and so on. These are 16-bit register values, but they can also store Floating-Point Data Handling (Modbus Master) (page 46). You can also write to these Modbus addresses using Modbus Function Codes 6 or 16.

Below is a sample command to read Modbus addresses 40001 to 40010 of node 1 on the Modbus network.

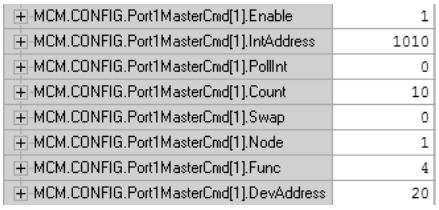
MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	1000
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	10
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func	3
MCM.CONFIG.Port1MasterCmd[0].DevAddress	0

Label	Description
Enable = 1	The module will send the command every time it goes through the command list.
IntAddress = 1000	Places the data read from the Slave device into the module at address 1000. IntAddress 1000 of the module memory will be copied into the tag **MCM.DATA.READDATA[0]**.
Count = 10	Reads 10 consecutive registers from the Slave device.
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 3	Issues Modbus Function Code 3 to Read Holding Registers.
DevAddress = 0	Function Code 3, DevAddress of 0 will read address 40001 Along with a count of 10, this command reads 40001 to 40010.

2.3.2 Read Input Registers 3x (Modbus Function Code 4)

Like the 4x holding registers, 3x input registers are used for reading analog values that are 16-bit register values. You can also use these registers to store Floating-Point Data Handling (Modbus Master) (page 46). Unlike the 4x registers, 3x registers are Read Only.

Below is a sample command to read Modbus addresses 30021 to 30030 of node 1 on the Modbus network.



MCM.CONFIG.Port1MasterCmd[1].Enable	1
MCM.CONFIG.Port1MasterCmd[1].IntAddress	1010
MCM.CONFIG.Port1MasterCmd[1].PollInt	0
MCM.CONFIG.Port1MasterCmd[1].Count	10
MCM.CONFIG.Port1MasterCmd[1].Swap	0
MCM.CONFIG.Port1MasterCmd[1].Node	1
MCM.CONFIG.Port1MasterCmd[1].Func	4
MCM.CONFIG.Port1MasterCmd[1].DevAddress	20

Label	Description
Enable = 1	The module will send the command every time it goes through the command list.
IntAddress = 1010	Places the data read from the Slave device into the module at address 1010. IntAddress 1010 of the module memory will be copied into the tag **MCM.DATA.READDATA[10].**
Count = 10	Reads 10 consecutive registers from the Slave device.
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 4	Issues Modbus Function Code 4 to Read Input Registers.
DevAddress =20	Function Code 4 DevAddress of 20 will read address 30021 Along with a count of 10, this command reads 30021 to 30030.

2.3.3 Read Coil Status 0x (Modbus Function Code 1)

Modbus Function Code 1 reads the Coils addressed at 0001 to 9999 from a Slave device. These are bit values that are read using Modbus Function Code 1, and can be written to using Function Code 5 or 15. Within a Slave device, this is an individual bit value. Thus, the IntAddress field must be defined down to the bit level within your MasterCmd.

Below is a sample command to read Modbus addresses 0321 to 0480 of node 1 on the Modbus network.

MCM.CONFIG.Port1MasterCmd[2].Enable	1
MCM.CONFIG.Port1MasterCmd[2].IntAddress	16320
MCM.CONFIG.Port1MasterCmd[2].PollInt	0
MCM.CONFIG.Port1MasterCmd[2].Count	160
MCM.CONFIG.Port1MasterCmd[2].Swap	0
MCM.CONFIG.Port1MasterCmd[2].Node	1
MCM.CONFIG.Port1MasterCmd[2].Func	1
MCM.CONFIG.Port1MasterCmd[2].DevAddress	320

Label	Description
Enable = 1	The module will send the command every time it goes through the command list.
IntAddress = 16320	Places the data read from the Slave device into the module at address 16320. IntAddress 16320 of the module memory will be copied into the tag **MCM.DATA.READDATA[20]** because 16320 represents a bit address within the memory of the MVI56E-MCM module (16320 / 16 = register 1020).
Count = 160	Reads 160 consecutive bits from the Slave device.
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 1	Issues Modbus Function Code 1 to Read Coils.
DevAddress = 320	Function Code 1, DevAddress of 320 will read address 0321 Along with a count of 160, this command reads 0321 to 0480.

2.3.4 Read Input Status 1x (Modbus Function Code 2)

Use this command to read Input Coils from a Slave device. These are single bit addresses within a Modbus Slave device. Unlike Coils 0xxx, the Input Coils are Read Only values and cannot be written to by a Modbus Master device. Also like the Coils 0xxx, the IntAddress field of this command is defined down to the bit level within the module memory.

Below is a sample command to read Modbus addresses 10081 to 10090 of node 1 on the Modbus network.

MCM.CONFIG.Port1MasterCmd[3].Enable	1
MCM.CONFIG.Port1MasterCmd[3].IntAddress	16480
MCM.CONFIG.Port1MasterCmd[3].PollInt	0
MCM.CONFIG.Port1MasterCmd[3].Count	16
MCM.CONFIG.Port1MasterCmd[3].Swap	0
MCM.CONFIG.Port1MasterCmd[3].Node	1
MCM.CONFIG.Port1MasterCmd[3].Func	2
MCM.CONFIG.Port1MasterCmd[3].DevAddress	80

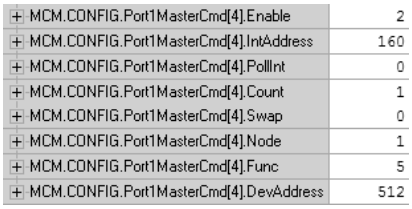
Label	Description
Enable = 1	The module will send the command every time it goes through the command list.
IntAddress = 16480	Places the data read from the Slave device into the module at address 16480. IntAddress 16480 of the module memory will be copied into the tag **MCM.DATA.READDATA[30]** (bit16480 / 16 = register 1030).
Count = 16	Reads 16 consecutive registers from the Slave device.

Label	Description
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 2	Issues Modbus Function Code 2 to Read Input Coils.
DevAddress = 80	Function Code 2, DevAddress of 80 will read address 10081 Along with a count of 16, this command reads 10081 to 10096.

2.3.5 Force (Write) Single Coil 0x (Modbus Function Code 5)

Used to write a Coil of a Slave device, these are single bit addresses within a Modbus Slave device. The IntAddress field of this command is defined down to the bit level within the module memory, and should come from an area of memory that has been defined within the **MCM.DATA.WRITEDATA** area (this is configured within **MCM.CONFIG.MODDEF.**

Below is a sample command to write Modbus addresses 0513 of node 1 on the Modbus network, only when the data associated with the IntAddress has changed.



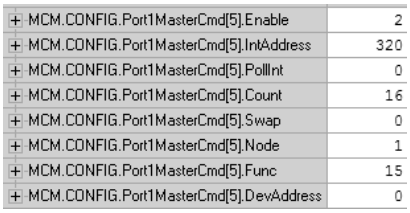
Label	Description
Enable = 2	The module will send the command only when the data within the IntAddress field of the module has changed.
IntAddress = 160	Will write the data to the Slave device when the value at WriteData[10].0 has changed. Because this is a bit level command, the IntAddress field must be defined down to the bit level.
Count = 1	Will write a single bit to the device (Function Code 5 will 1 support a count of 1).
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 5	Issues Modbus Function Code 5 to write a single coil.
DevAddress = 512	Function Code 5, DevAddress of 512 will read address 0513

2.3.6 Force (Write) Multiple Coils 0x (Modbus Function Code 15)

Use this function code to write multiple Coils in the 0xxx address range. This function code sets multiple Coils within a Slave device using the same Modbus command. Not all devices support this function code. Refer to your Slave device documentation before implementing this function code.

This function code will also support the Enable code of 2, to write the data to the Slave device only when the data associated within the IntAddress field of the module has changed. The IntAddress is once again defined down to the bit level as a Function Code 15 is a bit level Modbus function.

Below is a sample command to write Modbus addresses 0001 to 0016 of node 1 on the Modbus network.

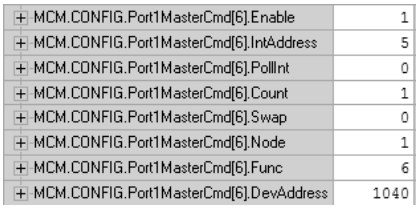


Label	Description
Enable = 2	The module will send the command to the Slave device only when the data associated within the IntAddress of the MVI56E-MCM module memory has changed.
IntAddress = 320	Writes the data in bit 320 of the module memory to the Slave device. Based on the **MCM.CONFIG.MODDEF** setting, this would be the data in **MCM.DATA.WRITEDATA[20].0** to **[20].15** in the ladder logic.
Count = 16	Writes 16 consecutive bits to the Slave device.
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 15	Issues Modbus Function Code 15 to write multiple coils.
DevAddress = 0	Function Code 15, DevAddress of 0 will read address 0001 Along with a count of 16, this command writes to 0001 to 0016.

2.3.7 Preset (Write) Single Register 4x (Modbus Function Code 6)

Used to write to Modbus Holding Registers 4xxxx, this function code will write a single register to the Slave device. The Enable code can be set to a value of 1 for a continuous write, or a value of 2 to write the data to the Slave device only when the data associated with the IntAddress field has changed.

Below is a sample command to write Modbus addresses 41041 of node 1 on the Modbus network.



Label	Description
Enable = 1	The module will send the command every time it goes through the command list.
IntAddress = 5	Writes the data from address 5 of the module memory to the Slave device. Based on the **MCM.CONFIG.MODDEF** configuration, this will take the data from **MCM.DATA.WRITEDATA[5]** and write that information out to the Slave device.
Count = 1	Writes 1 register (16-bit) to the Slave device.

Label	Description
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 2	Issues Modbus Function Code 6 to write a single register.
DevAddress = 1040	Function Code 6, DevAddress of 1040 will write to address 41041 of the Modbus Slave device.

2.3.8 Preset (Write) Multiple Registers 4x (Modbus Function Code 16)

Used to write to Modbus Holding Registers 4xxxx, this function code will write multiple registers to the Slave device. The Enable code can be set to a value of 1 for a continuous write, or a value of 2 to write the data to the Slave device only when the data associated with the IntAddress field has changed.

Below is a sample command to write Modbus addresses 41051 to 41060 of node 1 on the Modbus network.

MCM.CONFIG.Port1MasterCmd[7].Enable	2
MCM.CONFIG.Port1MasterCmd[7].IntAddress	30
MCM.CONFIG.Port1MasterCmd[7].PollInt	0
MCM.CONFIG.Port1MasterCmd[7].Count	10
MCM.CONFIG.Port1MasterCmd[7].Swap	0
MCM.CONFIG.Port1MasterCmd[7].Node	1
MCM.CONFIG.Port1MasterCmd[7].Func	16
MCM.CONFIG.Port1MasterCmd[7].DevAddress	1050

Label	Description
Enable = 2	The module will send the command only when the data associated with the IntAddress of the module has changed.
IntAddress =30	Writes the data from Internal Address 30 of the module memory to the Slave device. Based on the **MCM.CONFIG.MODDEF** configuration, this will write the data from **MCM.DATA.WRITEDATA[30] TO [39]** to the Slave device.
Count = 10	Writes 10 consecutive registers to the Slave device.
Node = 1	Issues the Modbus command to node 1 on the network.
Func = 16	Issues Modbus Function Code 16 to write Holding Registers.
DevAddress = 1050	Function Code 16, DevAddress of 1050 will write address 41051. Along with a count of 10, this command writes 41051 to 41060 of the Slave device.

2.4 Floating-Point Data Handling (Modbus Master)

In many applications, it is necessary to read or write floating-point data to the Slave device. The sample program only provides an INT array for the ReadData and Write Data array (16-bit signed integer value). In order to read/write floating-point data to and from the Slave device, you must add additional ladder to handle the conversion of the data to a REAL data type within the ControlLogix processor. This is very easy to accomplish.

The following topics show how to read or write data to a Slave device. These topics also show when to use the Float Flag and Float Start parameters within the module configuration. For all applications, floating-point data can be read from a device without any changes to the Float Flag and Float Start parameters. You only need to configure these parameters to issue a Write command to a device that uses a single Modbus address, such as 47001, to represent a single floating-point value.

2.4.1 Read Floating-Point Data

Here is the addressing of a Slave device, with a parameter "Energy Consumption" that is shown as two registers 40257 and 40258.

Value			Description	Type
40257	--------	KWH	Energy Consumption	Float, lower 16 bits
40258		KWH	Energy Consumption	Float, upper 16 bits

To issue a Read command to this parameter, use the following configuration.

Parameter	Value	Description
Enable	1	Sends the command every time through the command list.
IntAddress	1000	Places data at address 1000 of the module memory. Based on the configuration in ModDef this will put the data at the tag **MCM.DATA.READDATA[0]**.
PollInt	0	No delay for this command.
Count	2	Reads 2 consecutive registers from the Slave device. These 2 Modbus registers will make up the "Energy Consumption" floating-point value.
Swap	0	**Swap Code** / **Description** 0 – None - No Change is made in the byte ordering (1234 = 1234) 1 – Words - The words are swapped (1234=3412) 2 – Words & Bytes - The words are swapped then the bytes in each word are swapped (1234=4321) 3 – Bytes - The bytes in each word are swapped (1234=2143)
Node	1	Sends the command to Node #1.
Func	3	Issues a Modbus Function Code 3 to "Read Holding registers."
DevAddress	256	Along with the Function Code 3, DevAddress 256 will read Modbus address 40257 of the Slave device.

Along with the Function Code 3, DevAddress 256 will read Modbus address 40257 of the Slave device. The above command will read 40257 and 40258 of the Modbus Slave #1 and place that data in **MCM.DATA.READDATA[0]** and **[1].**

Within the controller tags section of the ControlLogix processor, it is necessary to configure a tag with the data type of "REAL" as shown in the following illustration.

[+]	Energy_Consumption	REAL[1]	Float

Copy data from the **MCM.DATA.READDATA[0]** and **[1]** into the tag **ENERGY_CONSUMPTION** that has a data type of REAL. Use a **COP** statement within the ladder logic. Here is an example.

Because the tag **MCM.DATA.READDATA[0]** should only be used within the above command, an unconditional COP statement can be used.

Notice the length of the COP statement is a value of 1. Within a Rockwell Automation processor, a COP statement will copy the required amount of "Source" values to fill the "Dest" tag for the Length specified.

Therefore, the above statement will copy ReadData[0] and [1] to fill the 32 bits required for the tag "Energy_Consumption".

Note: Do not use a MOV statement. A MOV will convert the data from the Source register to the destination register data type. This would create a data casting statement and will result in the loss or corruption of the original data.

2.4.2 Read Multiple Floating-Point Registers

The following table is an example to read Multiple Floating-Point values and device addresses. The table shows 7 consecutive floating-point values (14 Modbus addresses).

Value		Description	Type
40261	KW	Demand (power)	Float. upper 16 bits
40263	VAR	Reactive Power	Float. upper 16 bits
40265	VA	Apparent Power	Float. upper 16 bits
40267		Power Factor	Float. upper 16 bits
40269	VOLTS	Voltage, line to line	Float. upper 16 bits
40271	VOLTS	Voltage, line to neutral	Float. upper 16 bits
40273	AMPS	Current	Float. upper 16 bits

Configure the command to read these 7 floats as follows.

MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	1000
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	14
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func	3
MCM.CONFIG.Port1MasterCmd[0].DevAddress	260

Configure an array of 7 floats within the ControlLogix processor as shown in the following illustration.

MCM_Float_Data			REAL[7]	Float

The following **COP** statement will copy the data from **MCM.DATA.READDATA[0] TO [13]** into the array **MCM_FLOAT_DATA[0] TO [6].**

The "Length" parameter is set to the number of Floating-Point values that must be copied from the **MCM.DATA.READDATA** array.

2.4.3 Write Floats to Slave Device

To issue a Write command to Floating-Point addresses, use the configuration in the following table. The table describes the Modbus Map for the Slave device.

Value		Description	Type
40261	KW	Demand (power)	Float. upper 16 bits
40263	VAR	Reactive Power	Float. upper 16 bits
40265	VA	Apparent Power	Float. upper 16 bits
40267		Power Factor	Float. upper 16 bits
40269	VOLTS	Voltage, line to line	Float. upper 16 bits
40271	VOLTS	Voltage, line to neutral	Float. upper 16 bits
40273	AMPS	Current	Float. upper 16 bits

You must use a **COP** statement to copy the data from floating-point data tags within the ControlLogix processor, into the **MCM.DATA.WRITEDATA** array used by the MVI56E-MCM module. Below is an example.

The length of this COP statement must now be 14. This will COP as many of the **MCM_FLOAT_DATA** values required to occupy the **MCM.DATA.WRITEDATA** array for a length of 14. This will take 7 registers, **MCM_FLOAT_DATA[0] TO [6],** and place that data into **MCM.DATA.WRITEDATA[0] TO [13].**

You must configure the command to write all 7 floats (14 Modbus addresses) as follows.

MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	0
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	14
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func	16
MCM.CONFIG.Port1MasterCmd[0].DevAddress	260

The above command will take the data from **MCM.DATA.WRITEDATA[0] TO [13]** and write this information to the Slave device node #1 addresses 40261 to 40274.

2.4.4 Read Floats with Single Modbus Register Address (Enron/Daniel Float)

Some Modbus Slave devices use a single Modbus address to store 32 bits of data. This type of data is typically referred to as Enron or Daniel Floating-Point.

A device that uses this addressing method may have the following Modbus Memory Map.

Address	Data Type	Parameter
47001	32 bit REAL	Demand
47002	32 bit REAL	Reactive Power
47003	32 bit REAL	Apparent Power
47004	32 bit REAL	Power Factor
47005	32 bit REAL	Voltage: Line to Line
47006	32 bit REAL	Voltage: Line to Neutral
47007	32 bit REAL	Current

This type of device uses one Modbus address per floating-point register. To read these values from the Slave device, configure the following command within the module.

MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	1000
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	7
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func	3
MCM.CONFIG.Port1MasterCmd[0].DevAddress	7000

Notice that the count is now set to a value of 7. Because the Slave device utilizes only 7 Modbus addresses, a count of 7 will cause the Slave to respond with 14 registers (28 bytes) of information.

Important: This command will still occupy 14 register within the **MCM.DATA.READDATA** array. You must not use addresses 1000 to 1013 in the IntAddress field for any other Modbus Master commands.

The **COP** statement for this type of data is the same as shown in Read Multiple Floating-Point Registers (page 48).

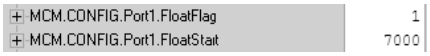
2.4.5 Write to Enron/Daniel Floats

To issue a Write command to Enron/Daniel Floats, use the Float Flag and Float Start parameters within the ModDef controller tags.

The following table describes the addresses that will be written to by the module.

Address	Data Type	Parameter
47001	32 bit REAL	Demand
47002	32 bit REAL	Reactive Power
47003	32 bit REAL	Apparent Power
47004	32 bit REAL	Power Factor
47005	32 bit REAL	Voltage: Line to Line
47006	32 bit REAL	Voltage: Line to Neutral
47007	32 bit REAL	Current

Configure the Float Start and Float Flag parameters as shown.



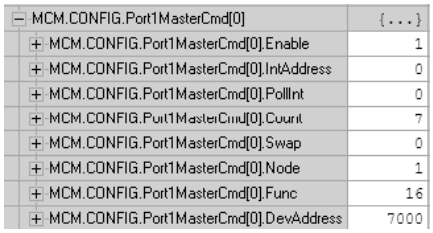
The Float Flag causes the module to use the FloatStart parameter to determine which DevAddress requires a write command to issue double the number of bytes.

With the above configuration, any DevAddress > 7000 is known to be floating-point data. Therefore, a count of 1 will send 4 bytes of data, instead of the normal 2 bytes of data to a non Enron/Daniel floating-point register.

1. First, copy the floating-point data from the ControlLogix processor into the **MCM.DATA.WRITEDATA** array used by the MVI56E-MCM module. Below is an example.

2. The length of this COP statement must now be 14. This will COP as many of the **MCM_FLOAT_DATA** values required to occupy the **MCM.DATA.WRITEDATA** array for a length of 14. This will take 7 registers, **MCM_FLOAT_DATA[0] TO [6],** and place that data into **MCM.DATA.WRITEDATA[0] TO [13].**

The following illustration shows the command required to write these 7 Floating-Point values.



Based on the IntAddress and the configuration within the **MCM.CONFIG.MODDEF** section for WriteStartReg and WriteRegCount, the data from the tag **MCM.DATA.WRITEDATA[0] TO [6]** will be written to Modbus addresses 47001 to 47007 of the Slave device node #1.

Note: A swap code may be required to put the data in the proper format for the Slave device.

2.5 Command Control and Event Command

You can use Command Control and Event Commands in Modbus Master mode to change the command execution based on some conditions in ladder. The module goes through the command list sequentially. For example:

- The module executes **MCM.CONFIG.PORT1MASTERCMD[0]**
- After completing that command, it will then execute **MCM.CONFIG.PORT1MASTERCMD[1],** then **MCM.CONFIG.PORT1MASTERCMD[2],** and so on.

You can use Command Control and Event Command to issue a command at the top of the command queue, interrupting the regular command list execution.

You would typically use Command Control and Event Command to

- issue a reset to a device on a once a day basis
- poll for end of hour data
- issue special commands on the startup of a process or the changing of a batch

Important: Because these special command blocks will interrupt the normal polling list, you should use them sparingly, to avoid interrupting your normal data transfer. Make sure that the data to be written to the device contains the latest value from the WriteData tag that corresponds to the Command Control or Event Command.

2.5.1 Command Control

Command Control allows you to issue a command already defined in the Master command list (but disabled) and enable that command for a single pass. Command Control has a distinct advantage over Event Command in that it will still return an error code for that command as configured in **MCM.CONFIG.PORTX.CMDERRPTR.** Up to 6 commands may be enabled at the same time.

The following illustration shows how to configure Command Control using the **MCM.UTIL.CMDCONTROL** object in the ladder logic.

MCM.UTIL.CmdControl	{...}
MCM.UTIL.CmdControl.TriggerCmdCntrl	0
MCM.UTIL.CmdControl.NumberOfCommands	6
MCM.UTIL.CmdControl.PortNumber	1
MCM.UTIL.CmdControl.CommandIndex	{...}
MCM.UTIL.CmdControl.CommandIndex[0]	0
MCM.UTIL.CmdControl.CommandIndex[1]	1
MCM.UTIL.CmdControl.CommandIndex[2]	2
MCM.UTIL.CmdControl.CommandIndex[3]	3
MCM.UTIL.CmdControl.CommandIndex[4]	4
MCM.UTIL.CmdControl.CommandIndex[5]	5
MCM.UTIL.CmdControl.CmdsAddedToQueue	0
MCM.UTIL.CmdControl.CmdControlBlockID	0
MCM.UTIL.CmdControl.CmdCntrlPending	0

The following configuration will place 6 commands into the command queue. **MCM.CONFIG.PORT1MASTERCMD[0]** to **MCM.CONFIG.PORT1MASTERCMD[5]** will be enabled with this configuration. Error codes for each command are placed in the Error Status table.

Tag	Value	Description
TriggerCmdCntrl	1	1 will execute the Command Control
NumberOfCommands	6	Number of commands per block
PortNumber	1	MVI56E-MCM Port number (Master)
CommandIndex[0] to [324]	0 to 324	Stores the command index for Command Control block
CmdsAddedToQueue		Number of commands added to queue. This is the confirmation that the Command Control block has completed successfully
CmdControlBlockID		Temporary variable to calculate control block ID number
CmdCntrolPending		Aux. control command - prevents a second request before acknowledgement is received

Note: For RSLogix version 15 and lower, the ladder logic necessary for the successful execution of this block is contained in the _WriteControl ladder file, rung 4, and in the _ReadControl ladder file, rung 2.

2.5.2 Event Command

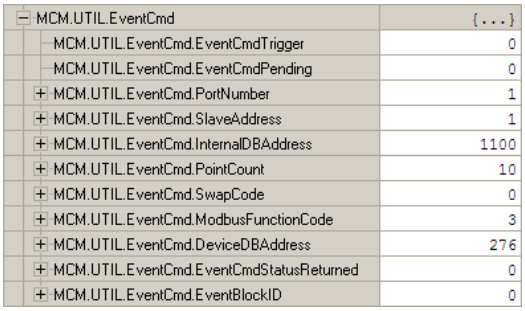
Event Command allows you to add commands directly to the command queue, interrupting the normal polling sequence of the module. Unlike Command Control, Event Commands do not return an error code into the location defined by the **MCM.CONFIG.PORTX.CMDERRPTR** value.

You do not need to define Event Commands in the regular command list. Event Command adds a command to the top of the MVI56E-MCM module's command queue that is not defined within the command list.

Within an Event Command block, you define a Modbus command to add to the queue.

Important: Because these special command blocks will interrupt the normal polling list, you should use them sparingly, to avoid interrupting your normal data transfer. Make sure that the data to be written to the Slave contains the latest value from the WriteData tag that corresponds to the Event Command.

The following illustration describes the structure of the EventCmd block.



Parameter	Value	Description
EventCmdTrigger	1	1 = trigger the Event Command
EventCmdPending		Used = EventCommand is executed once
PortNumber	1	Module Port # to send command out to
SlaveAddress	1	Modbus Slave ID command to be issued to
InternalDBAddress	1100	1100 will place the data read into MCM.DATA.ReadData[100]
PointCount	10	Consecutive register/bits to read or write with the command
SwapCode	0	Swap code used with command
ModbusFunctionCode	3	Function Code 3 is read 4xxxx holding registers
DeviceDBAddress	276	Address in the Slave device to read. With Function Code 3, DeviceDBAddress of 276, the module will read starting at address 40277 in the Slave device
EventCmdStatusReturned		Return value of 0 = Fail, 1 = Success
EventBlockID		Block ID number for the module to recognize the Event Command, Slave address, and Port number to send the command out

Note: For RSLogix version 15 and lower, the ladder logic used for the Event Command blocks is contained in _WriteControl rung 5 and _ReadControl rung 4 within the sample ladder file.
Note: Event Command blocks can only send 1 command to the command queue per block.
Note: Event Commands (like Command Control) take priority over commands in the normal command list.

3 Configuration as a Modbus Slave

In This Chapter

- Overview 57
- ModDef Settings 58
- Slave Configuration 62
- Floating-Point Data Handling (Modbus Slave) 63
- Read and Write Same Modbus Address (Pass Through) 65

3.1 Overview

When configuring the module as a Slave, you will be providing whoever is programming the Master side of the communications with a Modbus Memory Map.

Note: If you are using the Sample Ladder Logic, the transfer of data is already done.

Information that is to be read by the Modbus Master device will be placed in the **MCM.DATA.WRITEDATA** array as this will be pushed out to the module so that values from the ControlLogix processor can be read by the Modbus Master. Information that must be written to the ControlLogix processor from the Modbus Master device will be placed into the **MCM.DATA.READDATA** array.

To configure module as a Modbus Slave you must determine how much data you must transfer to and from the module, to the Modbus Master.

The sample ladder file is configured to transfer 600 16-bit registers in each direction. If more than that is required, please see Adjust the Input and Output Array Sizes (Optional) (page 27).

Find out if the Master can read from one Modbus address and write to another Modbus address, or, if the Master must use the same address to read and write data points.

If a Modbus command must bypass the read and write areas of the slave's memory area and send Modbus commands directly to another device on the Modbus network (for example, to a PLC), you must use Pass-Through mode. This allows the **MCM.DATA.WRITEDATA** array to be used for all data transfer to the Master. Because the data transfer of the MVI56E-MCM module cannot be bidirectional, when the Master issues a Modbus Write command in Pass-Through mode, the MVI56E-MCM module builds a special block of information. This block is then parsed by the ladder logic, and the value written from the Modbus Master is then updated in the **MCM.DATA.WRITEDATA** array.

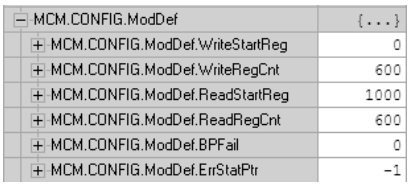
Note: You should only use Pass-Through mode when there is no other option, as there is a drawback to this mode that is not present in the standard mode.

Because the module must wait for the ladder logic to confirm receiving the new data from the Master, if the Master issues consecutive write commands, the module cannot process the second write command until it has finished with the first command. This will cause the module to respond with an error code of 6 (module busy) on the Modbus network.

3.2 ModDef Settings

To configure Modbus Slave mode, use the **MCM.CONFIG.MODDEF** settings.

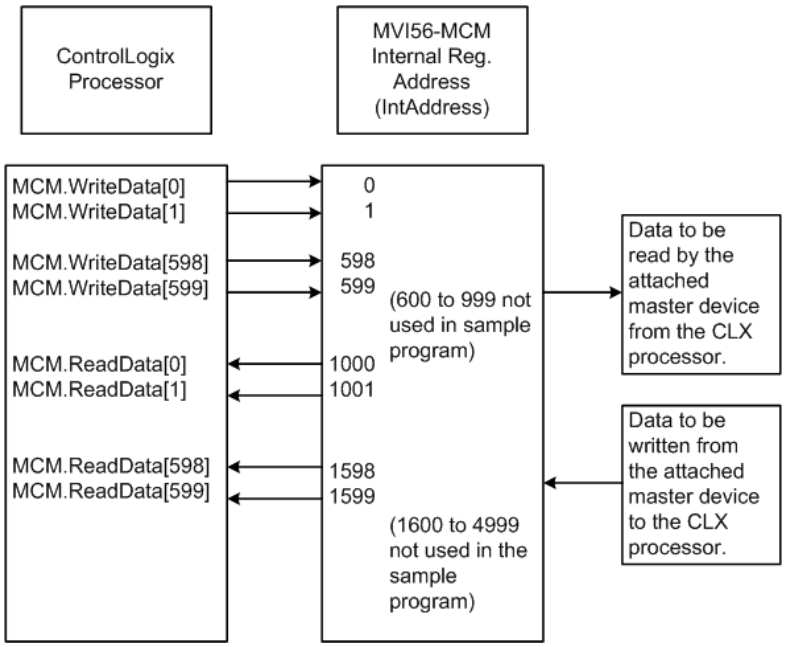
This section specifies which of the MVI56E-MCM module's 10,000 registers of memory to send from the ControlLogix processor to the MVI56E-MCM module (WriteData) and which registers to send from the MVI56E-MCM module to the ControlLogix processor (ReadData).



The **WRITESTARTREG** determines the starting register location for **WRITEDATA [0 TO 599]** and the **WRITEREGCNT** determines how many of the 10,000 registers to use for information to be written out to the module. The sample ladder file will configure 600 registers for Write Data, labeled **MCM.WRITEDATA[0 TO 599].**

Value	Description
WriteStartReg	Determines where in the 10,000 register module memory to place the data obtained from the ControlLogix processor from the WriteData tags.
WriteRegCnt	Sets how many registers of data the MVI56E-MCM module will request from the ControlLogix processor. Because the module pages data in blocks of 200 words, this number must be evenly divisible by 200.
ReadStartReg	Determines where in the 10,000 register module memory to begin obtaining data to present to the ControlLogix processor in the ReadData tags.
ReadRegCnt	Sets how many registers of data the MVI56E-MCM module will send to the ControlLogix processor. This value should also be a multiple of 200.
BPFail	Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network.
ErrStatPtr	This parameter places the STATUS data into the database of the module. This information can be read be the Modbus Master to know the status of the module.

With the sample configuration, the following is the layout of the tags and addressing.



The sample configuration values configure the module database for **WRITEDATA[0 TO 599]** to be stored in the module memory at register 0 to 599, and **READDATA[0 TO 599]** to be stored in the module memory at registers 1000 to 1599 as shown above.

3.2.1 Modbus Memory Map

Based on the configuration described above, below is the default Modbus address for the module. Each register within the module can be accessed as a 0xxx bit address, 1xxxx bit address, 3xxxx register address, or 4xxxx register address.

MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
0	0001 to 0016	10001 to 10016	30001	40001	WriteData[0]
1	0017 to 0032	10017 to 10032	30002	40002	WriteData[1]
2	0033 to 0048	10033 to 10048	30003	40003	WriteData[2]
3	0049 to 0064	10049 to 10064	30004	40004	WriteData[3]
4	0065 to 0080	10065 to 10080	30005	40005	WriteData[4]
5	0081 to 0096	10081 to 10096	30006	40006	WriteData[5]
6	0097 to 0112	10097 to 10112	30007	40007	WriteData[6]
7	0113 to 0128	10113 to 10128	30008	40008	WriteData[7]
8	0129 to 0144	10129 to 10144	30009	40009	WriteData[8]
9	0145 to 0160	10145 to 10160	30010	40010	WriteData[9]
10	0161 to 0176	10161 to 10176	30011	40011	WriteData[10]

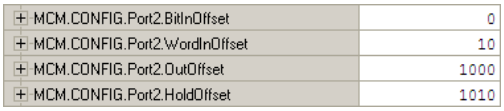
MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
50	0801 to 0816	10801 to 10816	30051	40051	WriteData[50]
100	1601 to 1616	11601 to 11616	30101	40101	WriteData[100]
200	3201 to 3216	13201 to 13216	30201	40201	WriteData[200]
500	8001 to 8016	18001 to 18016	30501	40501	WriteData[500]
598	9569 to 9584	19569 to 19584	30599	40599	WriteData[598]
599	9585 to 9600	19585 to 19600	30600	40600	WriteData[599]
600 to 999	N/A	N/A	N/A	N/A	Reserved
1000			31001*	41001	ReadData[0]
1001			31002*	41002	ReadData[1]
1002			31003*	41003	ReadData[2]
1003			31004*	41004	ReadData[3]
1004			31005*	41005	ReadData[4]
1005			31006*	41006	ReadData[5]
1006			31007*	41007	ReadData[6]
1007			31008*	41008	ReadData[7]
1008			31009*	41009	ReadData[8]
1009			31010*	41010	ReadData[9]
1010			31011*	41011	ReadData[10]
1050			31051*	41051	ReadData[50]
1100			31101*	41101	ReadData[100]
1200			31201*	41201	ReadData[200]
1500			31501*	41501	ReadData[500]
1598			31599*	41599	ReadData[598]
1599			31600*	41600	ReadData[599]

The above addressing chart will work with many Modbus applications. Values listed in the **READDATA** array for 31001 to 31600 are shown with an * beside them.

Although these are valid addresses, they will not work in the application. The Master must issue a Write command to the addresses that correspond to the **READDATA** array. For Modbus addresses 3xxxx these are considered Input registers, and a Modbus Master does not have a function code for this type of data.

3.2.2 Customizing the Memory Map

In some cases, the above memory map will not work for the application. Sometimes a Master must read bits starting at address 0001, and also read a register starting at 40001. With the memory map in this Modbus Memory Map (page 59), this is not possible, as **WRITEDATA[0]** is seen as both 0001 to 0016, and 40001. To accommodate this, you can customize the starting location within the module for each device using the parameters shown below.



Parameter	Value	Description
BitInOffset	0	Defines the starting address within the module for 1xxxx Modbus addressing. A value of 0 sets 10001 to 10016 as address 0 in the MVI56E-MCM module.
WordInOffset	10	Defines the starting address within the module memory for 3xxxx registers.
OutOffset	1000	Defines the starting address within the module for 0xxx coils.
HoldOffset	1010	Defines the starting address within the module for 4xxxx addressing.

Based on the configuration described above for the ModDef section of the module and the values specified for the offset parameters, below is the Modbus addressing map for the module.

MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
0		10001 to 10016			WriteData[0]
1		10017 to 10032			WriteData[1]
9		10145 to 10160			WriteData[9]
10		10161 to 10176	30001		WriteData[10]
11		10177 to 10192	30002		WriteData[11]
100		11601 to 11616	30091		WriteData[100]
200		13201 to 13216	30191		WriteData[200]
500		18001 to 18016	30491		WriteData[500]
598		19569 to 19584	30489		WriteData[598]
599		19585 to 19600	30490		WriteData[599]
600 to 999	N/A	N/A	N/A	N/A	Reserved
1000	0001 to 0016				ReadData[0]
1001	0017 to 0032				ReadData[1]
1009	0145 to 0160				ReadData[9]
1010	0161 to 0176			40001	ReadData[10]
1011	0177 to 0192			40002	ReadData[11]
1050	0801 to 0816			40041	ReadData[50]

MVI Address	0xxx	1xxxx	3xxxx	4xxxx	Tag Address
1100	1601 to 1616			40091	ReadData[100]
1200	3201 to 3216			40191	ReadData[200]
1500	8001 to 8016			40491	ReadData[500]
1598	9569 to 9584			40589	ReadData[598]
1599	9585 to 9600			40590	ReadData[599]

With the offset parameters listed above, the Modbus Master could read from coils 10001 to 10176 using the tags **MCM.DATA.WRITEDATA[0] TO [9].** The Master could also read from address 30001 to 30490, and the data contained in those Modbus addresses would come from the tags **MCM.DATA.WRITEDATA[10] TO [499]** within the ControlLogix program.

The Master could then write to coils addressing 0001 to 0160 and this data would reside within the ControlLogix program in tags **MCM.DATA.READDATA[0] TO [9].** The Master could then write to registers using Modbus addresses 40001 to 40590, and this information would reside in addresses **MCM.DATA.READDATA[10] TO [599].**

Note: The offset parameter only sets the starting location for the data. As shown above, if the Master issues a Write command to address 40001, the data will go into the ControlLogix processor at address **MCM.DATA.READDATA[10].**

Likewise, a Write To bit address 0161 will also change to address **MCM.DATA.READDATA[10].0** within the program. Be careful not to overlap your data. You may want leave additional registers/bits unused to allow for future expansion in the program.

3.3 Slave Configuration

Any parameters not mentioned in this section are not used when the module is configured as a Modbus Master.

Value	Description
Enabled	1= enable port, 0 = disable port
Type	**1= Modbus Slave Port** The module also supports a variety of Pass-Through modes. See Pass-Through Blocks (page 129) for more information.
FloatFlag	As a Slave, emulates Enron/Daniel style floats. See Floating-Point Data Handling (Modbus Slave) (page 63) for more information.
FloatStart	Register offset in message for floating data point. See Floating-Point Data Handling (Modbus Slave) (page 63) for more information.
Protocol	0 = Modbus RTU mode, 1 = Modbus ASCII mode
Baudrate	Sets the baud rate for the port. Valid values for this field are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 384 or 3840 (for 38,400 baud), 576 or 5760 (for 57,600 baud) and 115,1152, or 11520 (for 115,200 baud)
Parity	0 = None, 1 = Odd, 2 = Even
DataBits	8 = Modbus RTU mode, 8 or 7 = Modbus ASCII mode

Value	Description
StopBits	Valid values are 1 or 2
SlaveID	Valid values are 1 to 247

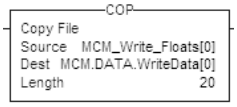
3.4 Floating-Point Data Handling (Modbus Slave)

In most applications, the use of floating-point data requires no special handling.

1. Copy the data to and from the MVI56E-MCM module with a tag configured as a data type REAL in the ControlLogix processor.

   Each floating-point value will occupy 2 registers on the Modbus network. Some Master devices use Enron or Daniel Float data. These types of floats require one Modbus register for each float in the module memory. If your Master requires this addressing, refer to the following section.
   For standard floating-point data handling, the following is an example of copying 10 floats to the module.
2. First, configure a tag within the ControlLogix processor.

3. Then configure a COP statement within the main routine to copy this tag to the module's **MCM.DATA.WRITEDATA** array.

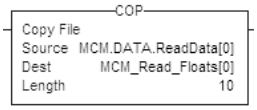


The length of the copy statement is determined by the Dest file size. To copy 10 floats from the MCM_Write_Floats array to the **MCM.DATA.WRITEDATA** array, the length of the COP statement must be set to a value of 20.

To copy data from the MVI56E-MCM module to a floating-point tag within the ControlLogix processor

1. Configure a tag within the ControlLogix processor as shown.

2. Then configure the COP statement to move data from the **MCM.DATA.READDATA** array, and over to the new tag **MCM_READ_FLOATS** tag as shown here.



Once again, the COP statement will take as many of the Source elements required to fill the Dest tag for the length specified. Therefore, the COP statement will take **MCM.DATA.READDATA[0] TO [19]** to fill the **MCM_READ_FLOATS[0] TO [9].**

3.4.1 Enron/Daniel Float Configuration

Sometimes it is necessary for the module to emulate Enron or Daniel floating-point addressing.

Copying the data to the **MCM.DATA.WRITEDATA** array and from the **MCM.DATA.READDATA** array is the same as described in the section above. The main difference is the addressing of the module.

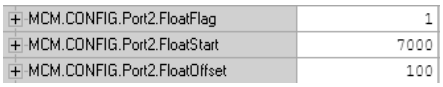
For example, an Enron Float device is required to access address 47001 for floating-point data, and each Modbus register would emulate a single float value (does not require 2 Modbus addresses for 1 float value).

A Master device requiring this type of addressing, would require that for every count of 1, the MVI56E-MCM module responds to the request message with 4 bytes (one 32-bit REAL) value.

To emulate this addressing, the module has the parameters **MCM.CONFIG.PORTX.FLOATFLAG, FLOATSTART,** and **FLOATOFFSET.**

Value	Description
FloatFlag	Tells the module to use the FloatStart and FloatOffset parameters listed below
FloatStart	Determines what starting address on the Modbus network to treat as floating-point data. A value of 7000 will signal the module that address 47001 on the Modbus network is the starting location for Modbus floating-point data. Every address will occupy 2 registers within the modules database
FloatOffset	Determines the address within the module to which to associate the data from the FloatStart section.

Here is a sample configuration for the module.



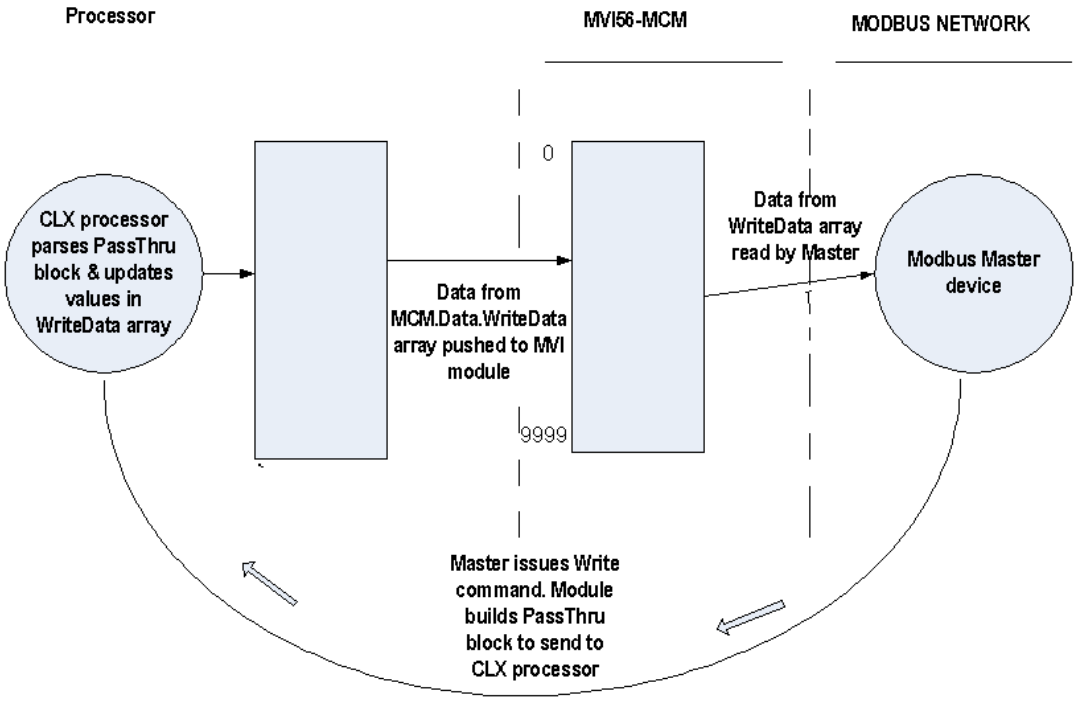
With the above configuration, this would be the addressing for the module.

Module Address	Modbus Address	Tag Address
100	47001	MCM.DATA.WriteData[100]
102	47002	MCM.DATA.WriteData[102]
104	47003	MCM.DATA.WriteData[104]
110	47006	MCM.DATA.WriteData[110]
120	47011	MCM.DATA.WriteData[120]
200	47051	MCM.DATA.WriteData[200]
300	47101	MCM.DATA.WriteData[300]
500	47201	MCM.DATA.WriteData[500]

3.5 Read and Write Same Modbus Address (Pass Through)

In some applications, the Modbus Master must be able to read and write to the same Modbus address within the module. This is not possible for normal Slave communication, as data can either be read from the WriteData array, or written to the ReadData array, but not both.

Pass Through mode allows the Modbus Master to bypass the module's internal memory, and then read and write directly to the processor, using only the WriteData array. The basic theory of pass through is that the ladder logic will constantly be updating values in the MVI56E-MCM module memory using the WriteData array. When the Master issues a Write command, the module will build a special block of data. This block of data is then presented to the ladder logic and then copied back into the WriteData array. The following illustration shows Pass Through operation of the module.


Processor
MVI56-MCM
MODBUS NETWORK
0
9999
CLX processor parses PassThru block & updates values in WriteData array
Data from MCM.Data.WriteData array pushed to MVI module
Data from WriteData array read by Master
Modbus Master device
Master issues Write command. Module builds PassThru block to send to CLX processor

Note: For RSLogix version 15 and lower, the ladder logic necessary for the successful execution of this block is contained in the subroutine _PassThru.
Pass Through should only be used when required. If a Master issues a Write command to the module, the module must build a special block of information. Then, it waits for confirmation from the ladder logic that the block has been processed.
Note: If the module is waiting for the block to be processed by the ladder, and the Master device issues another Write command, the module will return an Error Code of 6 (module busy). This error causes the ladder not to process data written by the Master.

4 Verify Communication

In This Chapter

- Verifying Master Communications 67
- Verify Slave Communications 73

There are several ways to verify that the MVI56E-MCM module is communicating with the processor and with the Modbus network.

- View the LED Status Indicators
- View the Module Status in the MVI56E-MCM Status Data Definition (page 152).
- View Diagnostics in Diagnostics and Troubleshooting (page 75)

4.1 Verifying Master Communications

The Modbus Master commands are configured, now it is time to verify that these commands are working correctly.

Within the MVI56E-MCM module, there are a couple of ways of checking to see if the commands that have been configured in the previous location are working correctly.

The most common, and detailed method of checking the communications is using the **MCM.CONFIG.PORTX.CMDERRPTR** parameter. This parameter will tell you the individual status of each command that is issued by the module. Another method is by checking the **MCM.STATUS.PRTXERRS** location for total commands issued, responses received, errors, and so on.

4.1.1 MVI56E-MCM Status Data Definition as a Master

This section contains a description of the members present in the **MCM.STATUS** object. This data is transferred from the module to the processor as part of each read block using the module's input image. Sample Ladder Logic will copy this information from the **LOCAL: X: I.DATA {OFFSET}** tag into the **MCM.STATUS** array.

Offset	Content	Description
202	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
203 to 204	Product Code	These two registers contain the product code of "MCM".
205 to 206	Product Version	These two registers contain the product version for the current running software.
207 to 208	Operating System	These two registers contain the month and year values for the program operating system.

Offset	Content	Description
209 to 210	Run Number	These two registers contain the run number value for the currently running software.
211	Port 1 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
212	Port 1 Command List Response	This field contains the number of Slave response messages received on the port.
213	Port 1 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
214	Port 1 Requests	This field contains the total number of messages sent from the port.
215	Port 1 Responses	This field contains the total number of messages received on the port.
216	Port 1 Errors Sent	This field contains the total number of message errors sent from the port.
217	Port 1 Errors Received	This field contains the total number of message errors received on the port.
218	Port 2 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
219	Port 2 Command List Response	This field contains the number of Slave response messages received on the port.
220	Port 2 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
221	Port 2 Requests	This field contains the total number of messages sent out the port.
222	Port 2 Responses	This field contains the total number of messages received on the port.
223	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
224	Port 2 Errors Received	This field contains the total number of message errors received on the port.
225	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
226	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
227	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
228	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
229	Command Block Count	This field contains the total number of command blocks received from the processor.
230	Error Block Count	This field contains the total number of block errors recognized by the module.
231	Port 1 Current Error	For a Master Port, this field contains the command index number of the most recently executed command that failed. To find what kind of error occurred, see the Command Error List entry for this command index number.

Offset	Content	Description
232	Port 1 Last Error	For a Master Port, this field contains the command index number of the previous most recently executed command that failed. To find what kind of error occurred, see the Command Error List entry for this command index number.
233	Port 2 Current Error	For a Master Port, this field contains the command index number of the most recently executed command that failed. To find what kind of error occurred, see the Command Error List entry for this command index number.
234	Port 2 Last Error	For a Master Port, this field contains the command index number of the previous most recently executed command that failed. To find what kind of error occurred, see the Command Error List entry for this command index number.

4.1.2 Command Error Codes

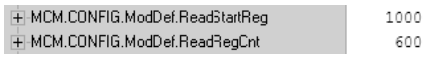
The MVI56E-MCM module will return an individual error code for every command configured within the **MCM.CONFIG.PORTXMASTERCMD** section. The location of these error codes are determined by the parameter **MCM.CONFIG.PORTX.CMDERRPTR.** This parameter determines where in the module's 10,000-register database the error codes for each command will be placed. The amount of error codes returned into the database is determined by the **MCM.CONFIG.PORTX.CMDCOUNT** parameter, therefore if the maximum number of commands have been selected (325), then 325 registers will be placed into the module memory.

Note: To use up to 325 commands, your MVI56E-MCM module needs to have firmware version 3.01 or higher, and your MVI56E-MCM Add-On Instruction needs to be version 2.8 or higher. Earlier versions support up to 100 commands.

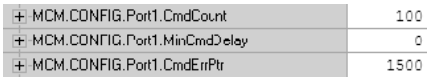
To be useful in the application, these error codes must be placed within the **MCM.DATA.READDATA** array.

Once again, the configuration in the **MCM.CONFIG.MODDEF** section for **READSTARTREG,** and **READREGCOUNT** determine which of the 10,000 registers will be presented to the ControlLogix processor and placed in the tag **MCM.DATA.READDATA** array.

Based on the sample configuration values for **READSTARTREG** and **READREGCNT,** this will be addresses 1000 to 1599 of the module memory. Below are the sample configuration values.



Based on these values shown above, a good place for the **MCM.CONFIG.PORTX.CMDERRPTR** is address 1500, as shown.



With the **CMDERRPTR** pointer set to address 1500 and the **CMDCOUNT** set to a value of 100, this will place your Command Error Data at addresses 1500 to 1599 of the module memory, and because of the before mentioned configuration of the **MCM.CONFIG.MODDEF READSTARTREG** and **READREGCNT** parameters, the command error data will be placed into the tags **MCM.DATA.READDATA[500] TO [599].**

Each command configured in the **MCM.CONFIG.PORTX.MASTERCMD** will occupy one register within the **READDATA** array. Based on the sample configuration values, the following table is true.

Error Code for Command	ReadData Location
MCM.CONFIG.Port1MasterCmd[0]	MCM.DATA.ReadData[500]
MCM.CONFIG.Port1MasterCmd[1]	MCM.DATA.ReadData[501]
MCM.CONFIG.Port1MasterCmd[2]	MCM.DATA.ReadData[502]
MCM.CONFIG.Port1MasterCmd[3]	MCM.DATA.ReadData[503]
MCM.CONFIG.Port1MasterCmd[4]	MCM.DATA.ReadData[504]
MCM.CONFIG.Port1MasterCmd[98]	MCM.DATA.ReadData[598]
MCM.CONFIG.Port1MasterCmd[99]	MCM.DATA.ReadData[599]

To know where to look for the error data, you need to know what the individual error codes are.

The following tables describe the possible error codes for the module:

Standard Modbus Protocol Errors

Code	Description
1	Illegal Function
2	Illegal Data Address
3	Illegal Data Value
4	Failure in Associated Device
5	Acknowledge
6	Busy, Rejected Message

The "Standard Modbus Protocol Errors" are error codes returned by the device itself. This means that the Slave device understood the command, but replied with an Exception Response, which indicates that the command could not be executed. These responses typically do not indicate a problem with port settings or wiring.

The most common values are Error Code 2 and Error Code 3.

Error Code 2 means that the module is trying to read an address in the device that the Slave does not recognize as a valid address. This is typically caused by the Slave device skipping some registers. If you have a Slave device that has address 40001 to 40005, and 40007 to 40010, you cannot issue a read command for addresses 40001 to 40010 (function code 3, DevAddress 0, Count 10) because address 40006 is not a valid address for this Slave.

Instead, try reading just one register, and see if the error code goes away. You can also try adjusting your DevAddress -1, as some devices have a 1 offset.

An Error Code of 3 is common on Modbus Write Commands (Function Codes 5,6,15, or 16). Typically, this is because you are trying to write to a parameter that is configured as read only in the Slave device, or the range of the data you are writing does not match the valid range for that device.

Refer to the documentation for your Slave device, or contact ProSoft Technical Support for more help with these types of error codes.

Module Communication Error Codes

Code	Description
-1	CTS modem control line not set before transmit
-2	Timeout while transmitting message
-11	Timeout waiting for response after request
253	Incorrect Slave address in response
254	Incorrect function code in response
255	Invalid CRC/LRC value in response

"Module Communication Errors" are generated by the MVI56E-MCM module, and indicate communication errors with the Slave device.

Error Code -11 indicates that the module is transmitting a message on the communications wire. However, it is not receiving a response from the addressed Slave. This error is typically caused by one or more of the following conditions.

- Parameter mismatch, for example the module is set for 9600 baud, Slave is set for 19,200, parity is set to none, Slave is expecting even, and so on.
- Wiring problem, for example the port jumper on the module is set incorrectly, or + and - lines on RS485 are switched)
- The Slave device is not set to the correct address, for example the Master is sending a command to Slave 1 and the Slave device is configured as device 10.

With a -11 error code, check all of the above parameters, wiring, and settings on the Slave device. Also make sure that you cycle power to the module, or toggle the **MCM.CONTROL.WARMBOOT** or **COLDBOOT** bit to transfer the values in the **MCM.CONFIG** array to the module.

Error codes of 253 to 255 typically indicate noise on RS485 lines. Make sure that you are using the proper RS485 cable, with termination resistors installed properly on the line. If termination resistors are installed, try removing them as they are usually only required on cable lengths of more than 1000 feet.

Command List Entry Errors

Code	Description
-41	Invalid enable code
-42	Internal address > maximum address
-43	Invalid node address (< 0 or > 255)
-44	Count parameter set to 0
-45	Invalid function code
-46	Invalid swap code

The above error codes indicate that the module has detected an error when parsing the command.

For all commands that have not been configured (all parameters set to a value of 0) you will receive an error code of -44. To remove this error code, you can change your **MCM.CONFIG.PORTX.CMDCOUNT** parameter to the number of commands that are actually configured, cycle power to the module, or toggle the **MCM.CONTROL.WARMBOOT** or **COLDBOOT** bit to transfer the new values to the module.

Transferring the Command Error List to the Processor

You can transfer the command error list to the processor from the module database. To place the table in the database, set the Command Error Pointer **(MCM.PORT1.CMDERRPTR)** parameter to the database location desired.

In the sample ladder, the **MCM.PORT1.CMDERRPTR** tag is set to a value of 1100. This will cause the error value of command 0 to be placed at database address 1100. Each command error value occupies one database word. The error value for command 1 will be in location 1101 and the remaining values in consecutive database locations.

To transfer this table to the processor, refer to Command Error Codes (page 69). Make sure that the Command Error table is in the database area covered by the Read Data **(MCM.MODDEF.READSTARTREG** and **MCM.MODDEF.READREGCNT).**

4.1.3 MCM Status Data

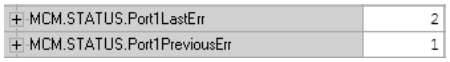
Status information can also be obtained from the MVI56E-MCM module by checking the **MCM.STATUS.PRTXERRS** location. Below is a sample.

MCM.STATUS.Prt1Errs	{...}
MCM.STATUS.Prt1Errs.CmdReq	1768
MCM.STATUS.Prt1Errs.CmdResp	1768
MCM.STATUS.Prt1Errs.CmdErr	0
MCM.STATUS.Prt1Errs.Requests	1768
MCM.STATUS.Prt1Errs.Responses	1768
MCM.STATUS.Prt1Errs.ErrSent	0
MCM.STATUS.Prt1Errs.ErrRec	0

If your system is working correctly, you will see **CMDREQ, CMDRESP, REQUESTS,** and **RESPONSES** all incrementing together. If you see that **CMDERR** is incrementing, determine what command is causing the error (using the error code defined in the previous Command Error Codes (page 69)) and correct the issue causing the error.

Note: This information is not as detailed as the individual error codes, but they can help to troubleshoot your application.

Also within the **MCM.STATUS** location is the parameters for Last Error and Previous Error, shown below.



This indicates the command index that last generated an error and does not indicate a command currently in error. In the above example, a value of 2 in **PORT1LASTERR** indicates that the last error was generated by **MCM.PORT1MASTERCMD[2].** This does not indicate that this command is currently in error. The value in **MCM.STATUS.PORT1PREVIOUSERR** indicates that before **MASTERCMD[2]** generated an error, **MCM.PORT1.MASTERCMD[1]** posted an error.

4.2 Verify Slave Communications

For verifying the communications to the module as a Slave you can monitor the **STATUS** tags under the **PRTXERRS** section.

Below is an example.

The **REQUESTS** field shows the number of request messages sent to the module as a Slave. The **RESPONSES** field shows how many times the module has responded to a request message from the Modbus Master.

4.2.1 MVI56E-MCM Status Data Definition as a Slave

This section contains a description of the members present in the **MCM.STATUS** object. This data is transferred from the module to the processor as part of each read block using the module's input image. Sample Ladder Logic will copy this information from the **LOCAL: X: I.DATA {OFFSET}** tag into the **MCM.STATUS** array.

Offset	Content	Description
202	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
203 to 204	Product Code	These two registers contain the product code of "MCM".

Offset	Content	Description
205 to 206	Product Version	These two registers contain the product version for the current running software.
207 to 208	Operating System	These two registers contain the month and year values for the program operating system.
209 to 210	Run Number	These two registers contain the run number value for the currently running software.
214	Port 1 Requests	This field contains the total number of messages sent from the port.
215	Port 1 Responses	This field contains the total number of messages received on the port.
216	Port 1 Errors Sent	This field contains the total number of message errors sent from the port.
217	Port 1 Errors Received	This field contains the total number of message errors received on the port.
221	Port 2 Requests	This field contains the total number of messages sent out the port.
222	Port 2 Responses	This field contains the total number of messages received on the port.
223	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
224	Port 2 Errors Received	This field contains the total number of message errors received on the port.
225	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
226	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
227	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
228	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
229	Command Block Count	This field contains the total number of command blocks received from the processor.
230	Error Block Count	This field contains the total number of block errors recognized by the module.
231	Port 1 Current Error	For a Slave Port, this field contains the value of the most recently returned error code.
232	Port 1 Last Error	For a Slave Port, this field contains the value of the previous most recently returned error code.
233	Port 2 Current Error	For a Slave Port, this field contains the value of the most recently returned error code.
234	Port 2 Last Error	For a Slave Port, this field contains the value of the previous most recently returned error code.

5 Diagnostics and Troubleshooting

In This Chapter

- Ethernet LED Indicators 75
- Clearing a Fault Condition 77
- Troubleshooting the LEDs 78
- Setting Up ProSoft Configuration Builder 78
- Connecting Your PC to the Module 83
- Downloading the Project to the Module 98
- Using the Diagnostics Menu in ProSoft Configuration Builder 99
- Reading Status Data from the Module 111
- Configuration Error Codes 111
- Connect to the Module's Web Page 113

The module provides information on diagnostics and troubleshooting in the following forms:

- LED status indicators on the front of the module provide information on the module's status.
- Status data contained in the module can be viewed in *ProSoft Configuration Builder* through the Ethernet port.
- Status data values are transferred from the module to the processor.

5.1 Ethernet LED Indicators

The Ethernet LEDs indicate the module's Ethernet port status as follows:

LED	State	Description
Data	OFF	Ethernet connected at 10Mbps duplex speed
	AMBER Solid	Ethernet connected at 100Mbps duplex speed
Link	OFF	No physical network connection is detected. No Ethernet communication is possible. Check wiring and cables.
	GREEN Solid or Blinking	Physical network connection detected. This LED must be ON solid for Ethernet communication to be possible.

5.1.1 Scrolling LED Status Indicators

The scrolling LED display indicates the module's operating status as follows:

Initialization Messages

Code	Message
Boot / DDOK	Module is initializing
Ladd	Module is waiting for required module configuration data from ladder logic to configure the Modbus ports
Waiting for Processor Connection	Module did not connect to processor during initialization ▪ Sample ladder logic or AOI is not loaded on processor ▪ Module is located in a different slot than the one configured in the ladder logic/AOI ▪ Processor is not in RUN or REM RUN mode
Last config: <date>	Indicates the last date when the module changed its IP address. You can update the module date and time through the module's web page, or with the MVI56E Optional Add-On Instruction.
Config P1/P2 <Modbus mode> <Port type> <Baud> <Parity> <Data bits> <Stop Bits> <RS Interface> <ID (Slave)> <Cmds: (Master)>	After power up and every reconfiguration, the module will display the configuration of both ports. The information consists of: ▪ Modbus mode: RTU/ASCII ▪ Port type: Master/Slave ▪ Baud: 115200 / 57600 / 38400 / 19200 / 9600/ 4800 / 2400 / 1200 / 600 / 300 ▪ Parity: None / Even / Odd ▪ Data bits: 7 / 8 ▪ Stop bits: 1 / 2 ▪ RS Interface: RS-232 / RS-422 / RS-485 ▪ ID: Slave Modbus Address ▪ Cmds: Configured Modbus Master Commands

Operation Messages

After the initialization step, the following message pattern will be repeated.

```
<Backplane Status> <IP Address> <Backplane Status> <Port Status>
```

Code	Message
<Backplane Status>	OK: Module is communicating with processor ERR: Module is unable to communicate with processor. For this scenario, the <Port Status> message above is replaced with "Processor faulted or is in program mode".
<IP Address>	Module IP address
<Port Status>	OK: Port is communicating without error Master/Slave Communication Errors: port is having communication errors. Refer to Diagnostics and Troubleshooting (page 75) for further information about the error.

5.1.2 Non-Scrolling LED Status Indicators

The non-scrolling LEDs indicate the module's operating status as follows:

LED Label	Color	Status	Indication
APP	Red or Green	OFF	The module is not receiving adequate power or is not securely plugged into the rack. May also be OFF during configuration download.
		GREEN	The MVI56E-MCM is working normally.
		RED	The most common cause is that the module has detected a communication error during operation of an application port. The following conditions may also cause a RED LED: ▪ The firmware is initializing during startup ▪ The firmware detects an on-board hardware problem during startup ▪ Failure of application port hardware during startup ▪ The module is shutting down ▪ The module is rebooting due to a ColdBoot or WarmBoot request from the ladder logic or Debug Menu
OK	Red or Green	OFF	The module is not receiving adequate power or is not securely plugged into the rack.
		GREEN	The module is operating normally.
		RED	The module has detected an internal error or is being initialized. If the LED remains RED for over 10 seconds, the module is not working. Remove it from the rack and re-insert it to restart its internal program.
ERR	Red		Not used.

5.2 Clearing a Fault Condition

Typically, if the OK LED on the front of the module turns RED for more than ten seconds, a hardware problem has been detected in the module or the program has exited.

To clear the condition, follow these steps:

1. Turn off power to the rack.
2. Remove the card from the rack.
3. Verify that all jumpers are set correctly.
4. If the module requires a Compact Flash card, verify that the card is installed correctly.
5. Re-insert the card in the rack and turn the power back on.
6. Verify correct configuration data is being transferred to the module from the ControlLogix controller.

If the module's OK LED does not turn GREEN, verify that the module is inserted completely into the rack. If this does not cure the problem, contact ProSoft Technology Technical Support.

5.3 Troubleshooting the LEDs

Use the following troubleshooting steps if problems occur when the module is powered up. If these steps do not resolve the problem, please contact ProSoft Technology Technical Support.

Processor Errors

Problem Description	Steps to take
Processor Fault	Verify the module is securely plugged into the slot that has been configured for the module in the I/O Configuration of RSLogix. Verify the slot location in the rack has been configured correctly in the ladder logic.
Processor I/O LED flashes	This indicates a problem with backplane communications. A problem could exist between the processor and any installed I/O module, not just the MVI56E-DNPNET. Verify all modules in the rack are configured correctly.

Module Errors

Problem Description	Steps to take
Module Scrolling LED display: <Backplane Status> condition reads ERR	This indicates that backplane transfer operations are failing. Connect to the module's Configuration/Debug port to check this. To establish backplane communications, verify the following items: ▪ The processor is in RUN or REM RUN mode. ▪ The backplane driver is loaded in the module. ▪ The module is configured for read and write data block transfer. ▪ The ladder logic handles all read and write block situations. ▪ The module is properly configured in the processor I/O configuration and ladder logic.
OK LED remains RED	The program has halted or a critical error has occurred. Connect to the communication port to see if the module is running. If the program has halted, turn off power to the rack, remove the card from the rack and re-insert the card in the rack, and then restore power to the rack.

5.4 Setting Up ProSoft Configuration Builder

ProSoft Configuration Builder (PCB) provides a convenient way to configure, diagnose, and troubleshoot your MVI56E-MCM module.

5.4.1 Installing ProSoft Configuration Builder

The ProSoft Configuration Builder (PCB) software is used to configure the module. You can find the latest version of the ProSoft Configuration Builder (PCB) on our web site: http://www.prosoft-technology.com, or you can install it from the ProSoft Solutions DVD. The installation filename contains the PCB version number. For example, **PCB_4.1.0.4.0206.EXE**.

If you are installing PCB from the ProSoft website:

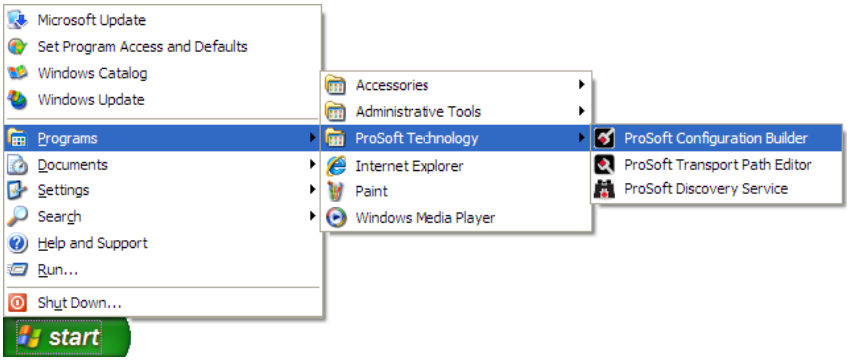
1. Open a browser window and navigate to http://www.prosoft-technology.com/pcb.
2. Click the download link for ProSoft Configuration Builder, and save the file to your Windows desktop.
3. After the download completes, double-click the file to install. If you are using Windows 7, right-click on the PCB installation file and click **RUN AS ADMINISTRATOR**. Follow the instructions that appear on the screen.
4. If you want to find additional software specific to your MVI56E-MCM, enter the model number into the website search box and press the Enter key.

If you are installing PCB from the ProSoft Solutions DVD:

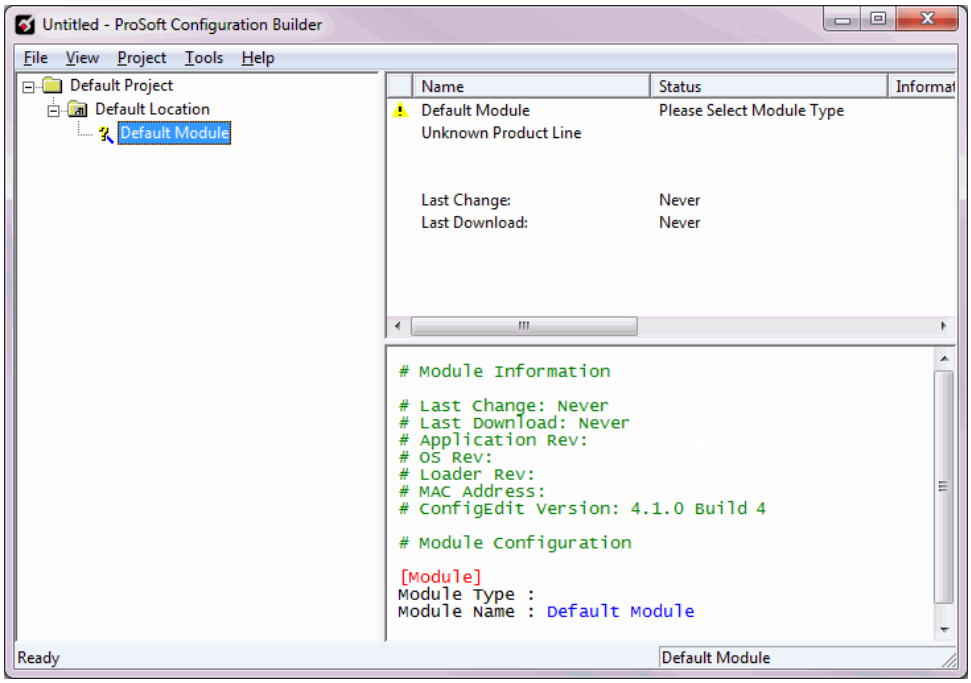
1. Insert the ProSoft Solutions DVD into your computer's DVD drive and wait for the ProSoft Installation program to start.
2. If the ProSoft Installation program does not start, open the Windows file Explorer, navigate to the DVD, and double-click on the ProSoft_DVD.exe file.
3. Navigate to the MVI56E-MCM selection using the **PLATFORM** and **PRODUCT** selections.
4. Click **PROSOFT CONFIGURATION BUILDER**.
5. Follow the instructions that appear on the screen.

5.4.2 Setting Up the Project

To begin, start **PROSOFT CONFIGURATION BUILDER** (PCB).



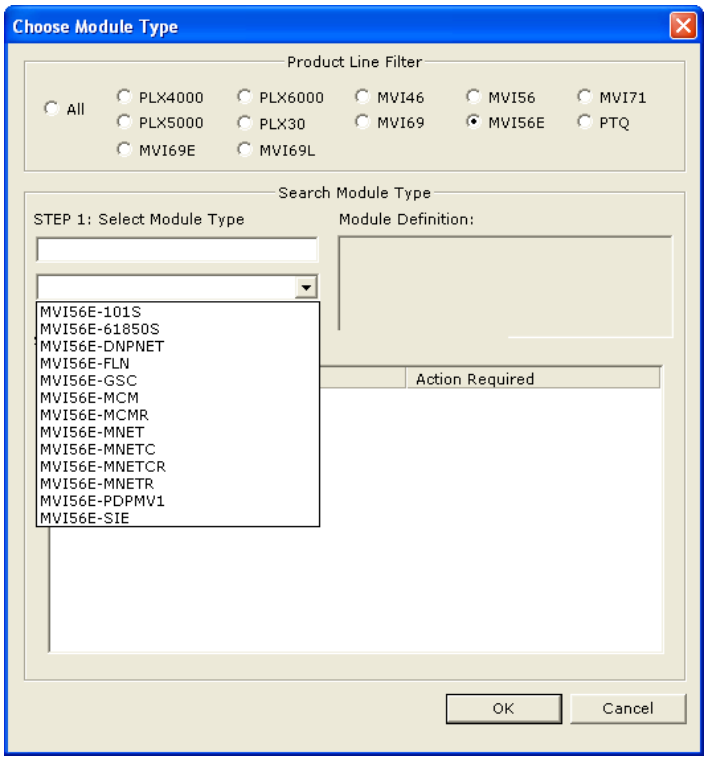
If you have used other Windows configuration tools before, you will find the screen layout familiar. *PCB*'s window consists of a tree view on the left, and an information pane and a configuration pane on the right side of the window. When you first start *PCB*, the tree view consists of folders for *Default Project* and *Default Location*, with a *Default Module* in the *Default Location* folder. The following illustration shows the *PCB* window with a new project.



Your first task is to add the MVI56E-MCM module to the project.

1 Use the mouse to select **DEFAULT MODULE** in the tree view, and then click the right mouse button to open a shortcut menu.

2 On the shortcut menu, select **CHOOSE MODULE TYPE**. This action opens the *Choose Module Type* dialog box.



3 In the *Product Line Filter* area of the dialog box, select **MVI56E**. In the *Select Module Type* dropdown list, select **MVI56E-MCM**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window.

5.4.3 Assigning an IP Address in the Project

In this step, you assign an IP address for the MVI56E-MCM module using ProSoft Configuration Builder. This becomes the permanent IP address for the module after you download the configuration to the module (refer to Downloading the Project to the Module (page 98)).

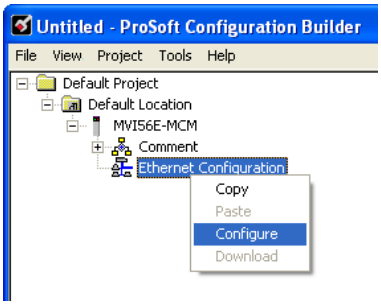
The module's default IP address is 192.168.0.250.

1 Determine the network settings for your module, with the help of your network administrator if necessary. You will need the following information:

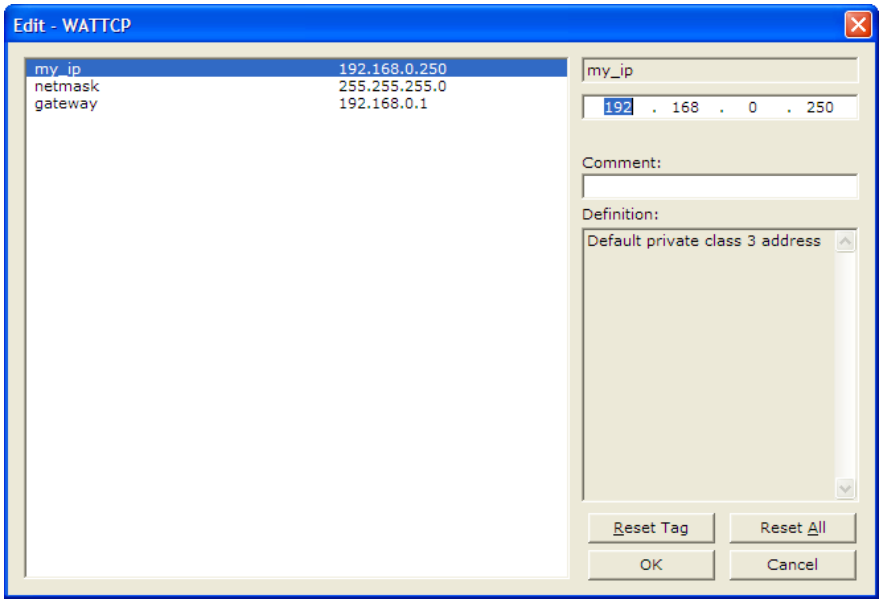
- IP address (fixed IP required) _____ . _____ . _____ . _____
- Subnet mask _____ . _____ . _____ . _____
- Gateway address _____ . _____ . _____ . _____

Note: The gateway address is optional, and is not required for networks that do not use a default gateway.

2 Start ProSoft Configuration Builder.
3 Select the **MVI56E-MCM** icon, and then click the **[+]** symbol to expand the MVI56E-MCM tree.
4 Right-click **ETHERNET CONFIGURATION** to open the shortcut menu.



5 On the shortcut menu, select **CONFIGURE**. This opens the *EDIT-WATTCP* dialog box.



6 Use this dialog box to enter the MVI56E-MCM module's permanent IP Address (**MY_IP**), subnet mask (**NETMASK**) and default gateway (**GATEWAY**).
7 Click **OK** to save the updated Ethernet configuration in the project.

5.5 Connecting Your PC to the Module

5.5.1 Download the IP Address through CIPconnect

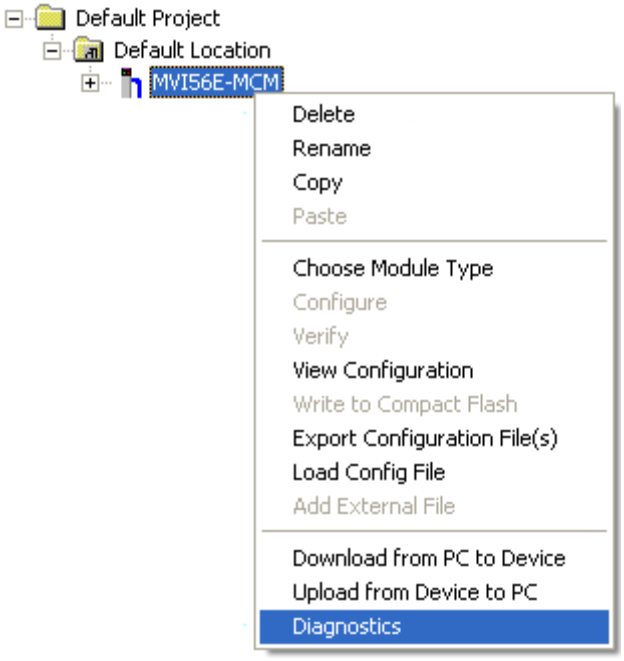
You can use CIPconnect® to connect a PC to the ProSoft Technology MVI56E-MCM module over Ethernet using Rockwell Automation's 1756-ENBT EtherNet/IP® module. This allows you to configure the MVI56E-MCM network settings and view module diagnostics from a PC. RSLinx is not required when you use CIPconnect. All you need are:

- The IP addresses and slot numbers of any 1756-ENBT modules in the path
- The slot number of the MVI56E-MCM in the destination ControlLogix chassis (the last ENBTx and chassis in the path).

If you do not have this information, you can still assign the IP address to the module (refer to Assigning a Temporary IP Address (page 94)).

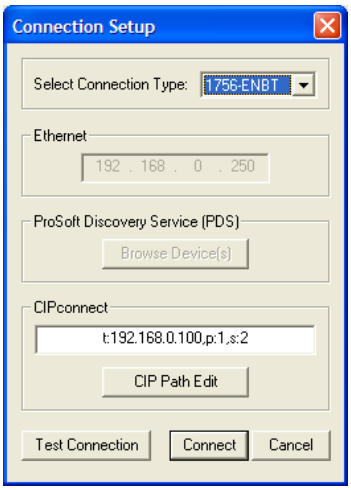
To use CIPconnect, follow these steps.

1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MCM** icon to open a shortcut menu.
2. On the shortcut menu, choose **DIAGNOSTICS.**

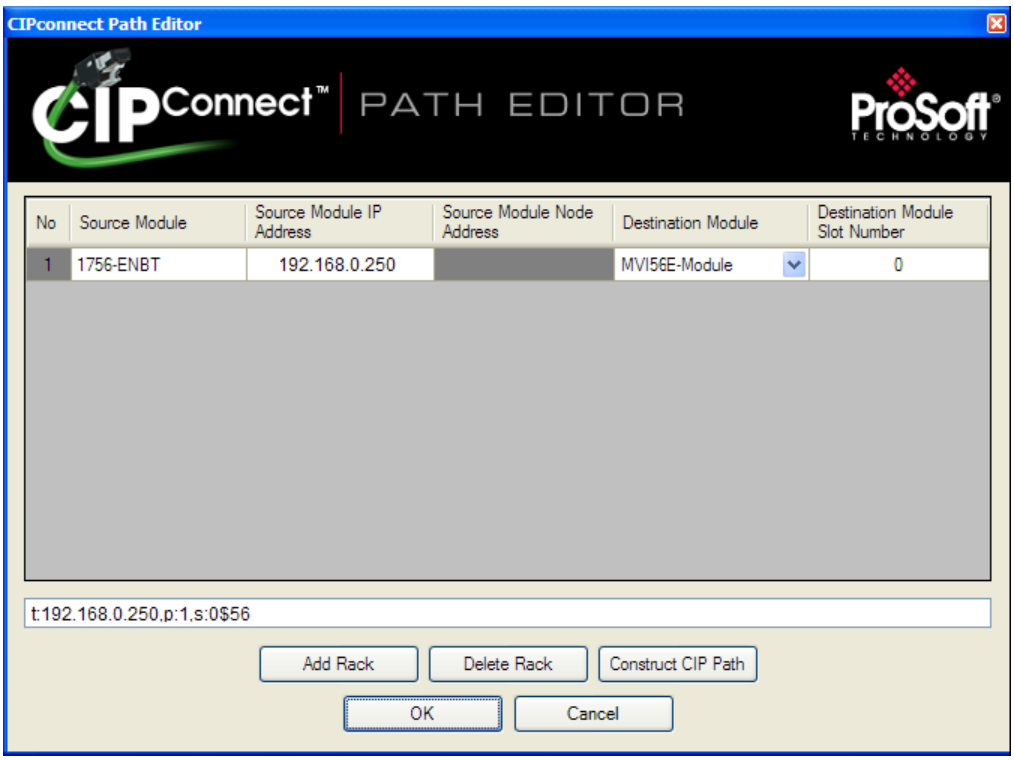


3. In the *Diagnostics* window, click the **SET UP CONNECTION** button.

4 In the *Select Connection Type* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.



5 Click **CIP PATH EDIT** to open the *CIPconnect Path Editor* dialog box.



The *CIPconnect Path Editor* allows you to define the path between the PC and the MVI56E-MCM module. The first connection from the PC is always a 1756-ENBT (Ethernet/IP) module.

Each row corresponds to a physical rack in the CIP path.

- If the MVI56E-MCM module is located in the same rack as the first 1756-ENBT module, select **RACK NO. 1** and configure the associated parameters.
- If the MVI56E-MCM is available in a remote rack (accessible through ControlNet or Ethernet/IP), include all racks (by using the **ADD RACK** button).

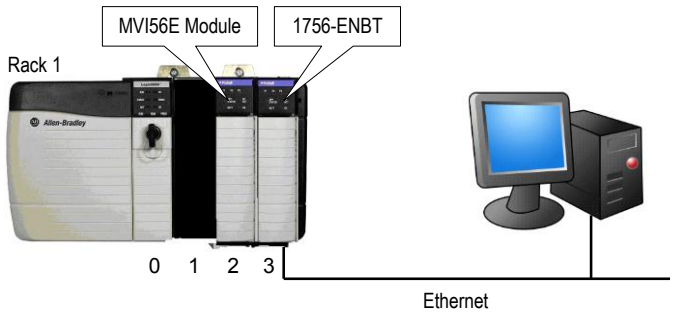
Parameter	Description
Source Module	Source module type. This field is automatically selected depending on the destination module of the last rack (1756-CNB or 1756-ENBT).
Source Module IP Address	IP address of the source module (only applicable for 1756-ENBT)
Source Module Node Address	Node address of the source module (only applicable for 1756-CNB)
Destination Module	Select the destination module associated to the source module in the rack. The connection between the source and destination modules is performed through the backplane.
Destination Module Slot Number	The slot number where the destination MVI56E module is located.

To use the CIPconnect Path Editor, follow these steps.

1. Configure the path between the 1756-ENBT connected to your PC and the MVI56E-MCM module.
   - If the module is located in a remote rack, add more racks to configure the full path.
   - The path can only contain ControlNet or Ethernet/IP networks.
   - The maximum number of supported racks is six.
2. Click **CONSTRUCT CIP PATH** to build the path in text format.
3. Click **OK** to confirm the configured path.

Example 1: Local Rack Application

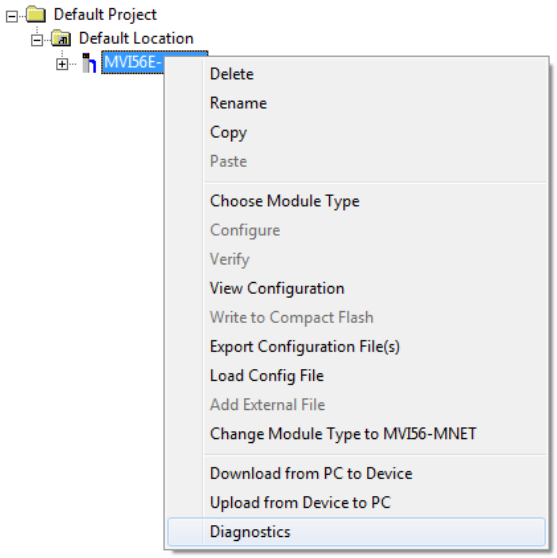
For this example, the MVI56E-MCM module is located in the same rack as the 1756-ENBT that is connected to the PC.



Rack 1

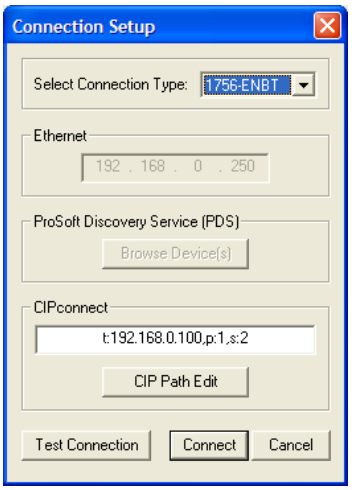
Slot	Module	Network Address
0	ControlLogix Processor	-
1	Any	-
2	MVI56E-MCM	-
3	1756-ENBT	IP=192.168.0.100

1. In *ProSoft Configuration Builder*, right-click the MVI56E-MCM icon to open a shortcut menu.
2. On the shortcut menu, choose **DIAGNOSTICS.**

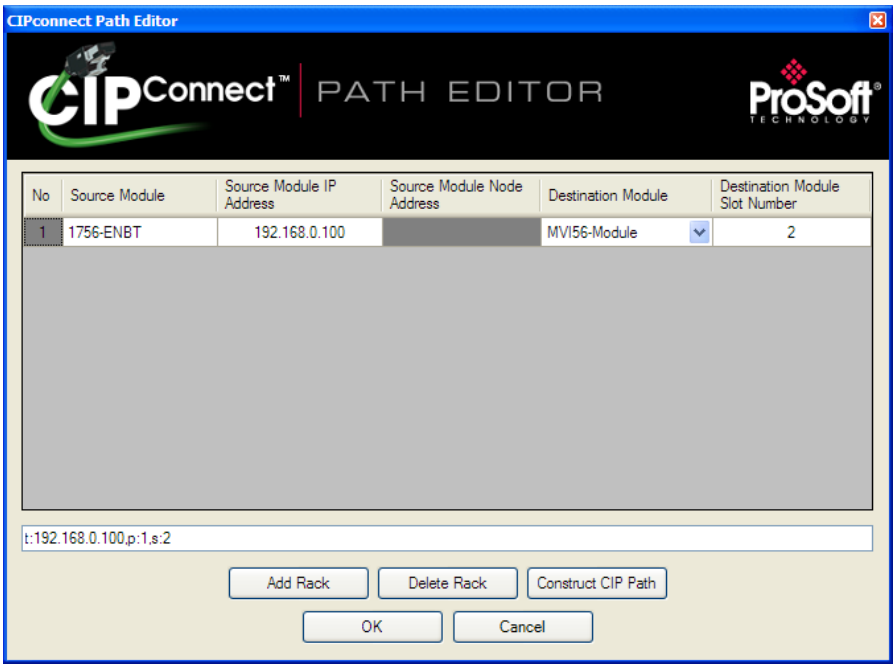


3. In the *Diagnostics* window, click the **SET UP CONNECTION** button.

4 In the *Select Connection Type* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.

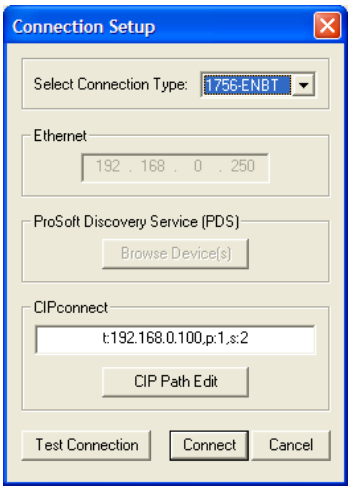


5 Configure the path as shown in the following illustration, and click **CONSTRUCT CIP PATH** to build the path in text format.

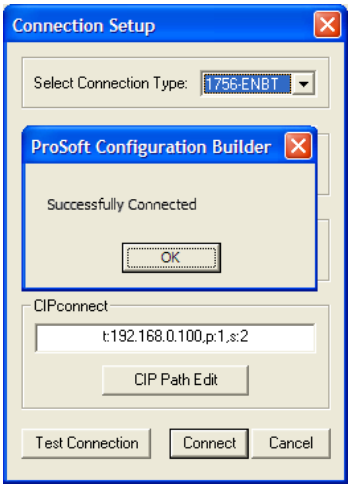


6 Click **OK** to close the *CIPconnect Path Editor* and return to the *Connection Setup* dialog box.

7 Check the new path in the *Connection Setup* dialog box.



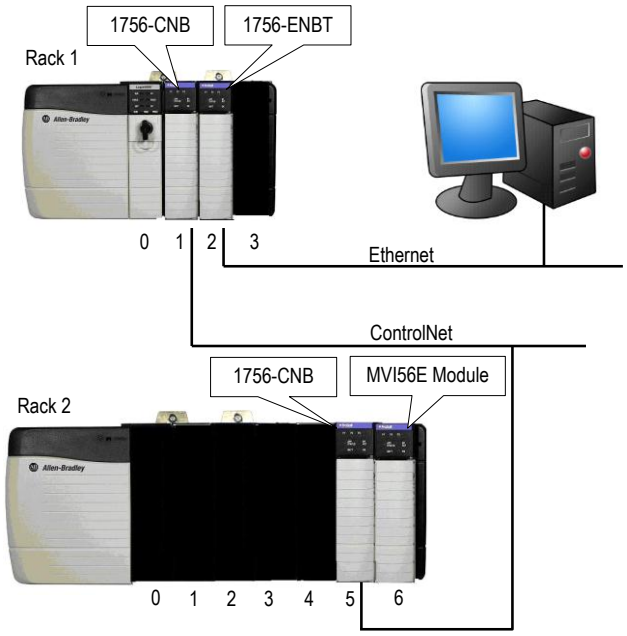
8 Click **TEST CONNECTION** to verify that the physical path is available. The following message should be displayed upon success.



9 Click **OK** to close the Test Connection pop-up and then click **CONNECT** to close the *Connection Set* up dialog box. The Diagnostics menu is now connected through CIPconnect.

Example 2: Remote Rack Application

For this example, the MVI56E-MCM module is located in a remote rack accessible through ControlNet, as shown in the following illustration.



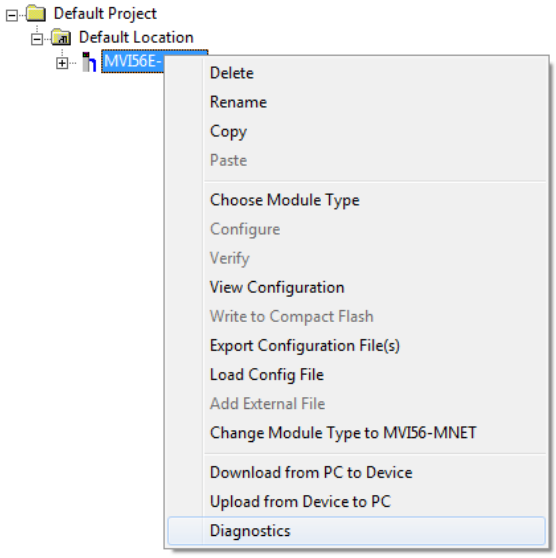
Rack 1

Slot	Module	Network Address
0	ControlLogix Processor	-
1	1756-CNB	Node = 1
2	1756-ENBT	IP=192.168.0.100
3	Any	-

Rack 2

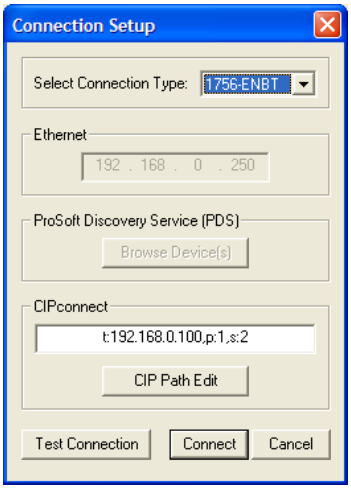
Slot	Module	Network Address
0	Any	-
1	Any	-
2	Any	-
3	Any	-
4	Any	-
5	1756-CNB	Node = 2
6	MVI56E-MCM	-

1 In *ProSoft Configuration Builder*, right-click the MVI56E-MCM icon to open a shortcut menu.
2 On the shortcut menu, choose **DIAGNOSTICS.**

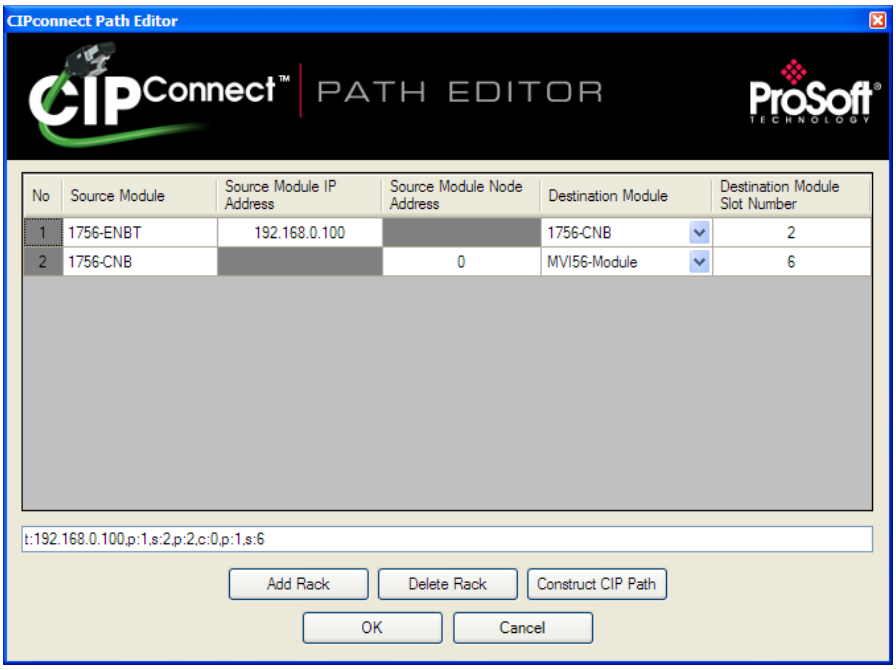


3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.

4 In the *Select Connection Type* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.

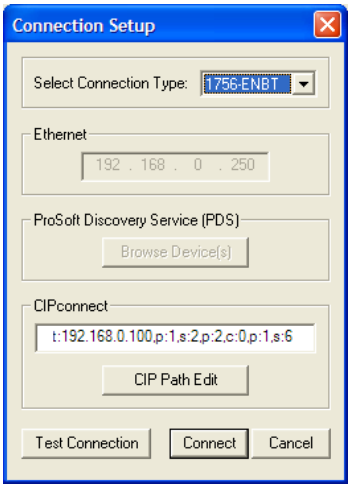


5 Configure the path as shown in the following illustration, and click **CONSTRUCT CIP PATH** to build the path in text format.

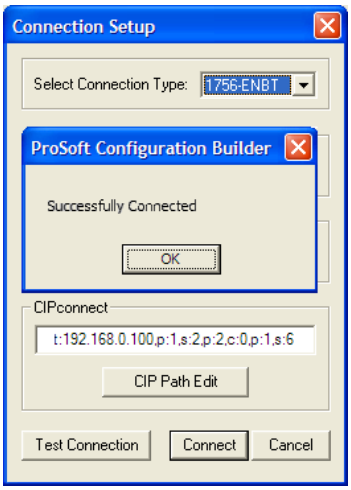


6 Click **OK** to close the *CIPconnect Path Editor* and return to the *Connection Setup* dialog box.

7 Check the new path in the *Connection Setup* dialog box.



8 Click **TEST CONNECTION** to verify that the physical path is available. The following message should be displayed upon success.



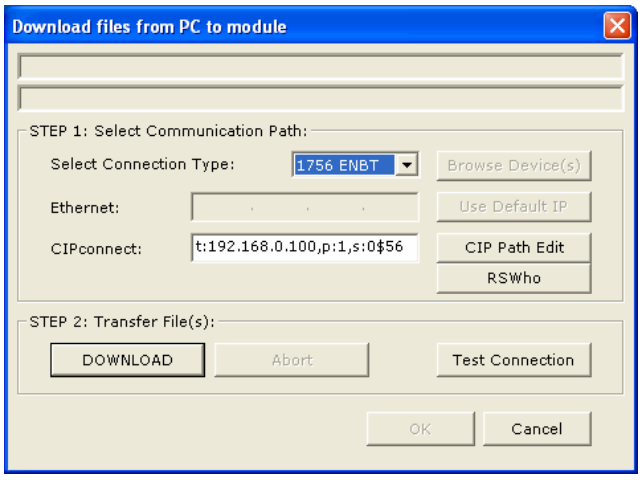
9 Click **OK** to close the Test Connection pop-up and then click **CONNECT** to close the *Connection Set* up dialog box. The Diagnostics menu is now connected through CIPconnect.

5.5.2 Using RSWho to Connect to the Module

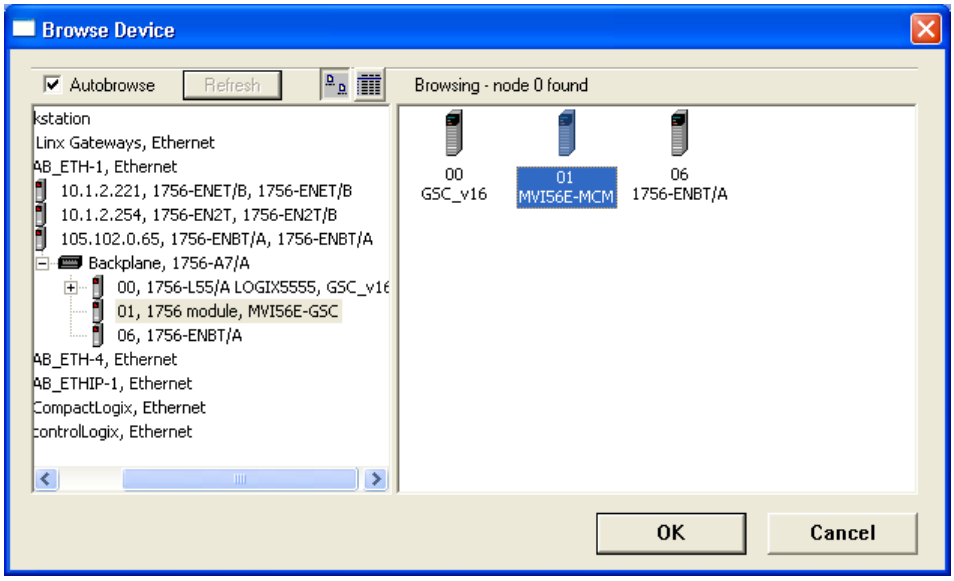
You need to have RSLinx installed on your PC to use this feature. You also need an ENBT module set up in the rack. For information on setting up the ENBT module, see Using CIPconnect to Connect to the Module.

1 In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MCM** module.
2 From the shortcut menu, choose **DOWNLOAD FROM PC TO DEVICE**.

3 In the *Download* dialog box, choose **1756 ENBT** from the *Select Connection Type* dropdown box.



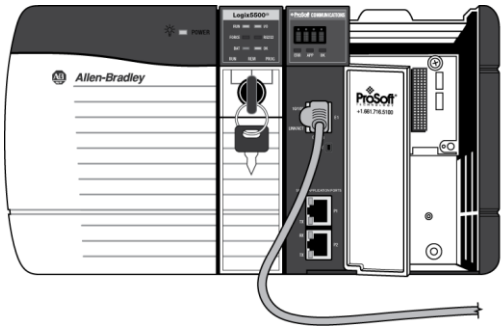
4 Click **RSWHO** to display modules on the network. The MVI56E-MCM module will automatically be identified on the network.



5 Select the module, and then click **OK**.

5.5.3 Connecting Your PC to the Module's Ethernet Port

With the module securely mounted, connect one end of the Ethernet cable to the **CONFIG (E1)** Port, and the other end to an Ethernet hub or switch accessible from the same network as your PC. Or, you can connect directly from the Ethernet Port on your PC to the **CONFIG (E1)** Port on the module.

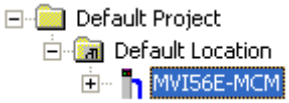


Assigning a Temporary IP Address

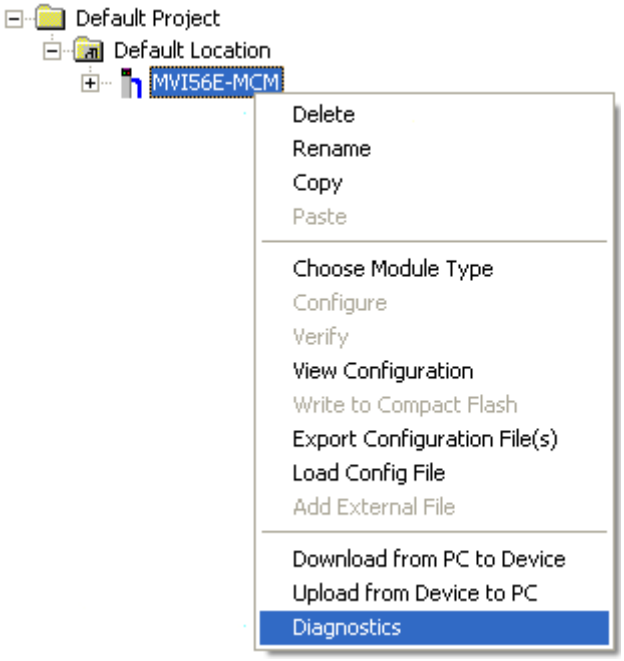
This procedure assigns a temporary IP address so that you can use the ProSoft Configuration Builder to download a configuration file containing the permanent IP address.

Important: *ProSoft Configuration Builder* locates MVI56E-MCM modules through UDP broadcast messages. These messages may be blocked by routers or layer 3 switches. In that case, *ProSoft Discovery Service* will be unable to locate the modules.
To use *ProSoft Configuration Builder*, arrange the Ethernet connection so that there is no router/ layer 3 switch between the computer and the module OR reconfigure the router/ layer 3 switch to allow routing of the UDP broadcast messages.

1 In the tree view in *ProSoft Configuration Builder*, select the **MVI56E-MCM** module.

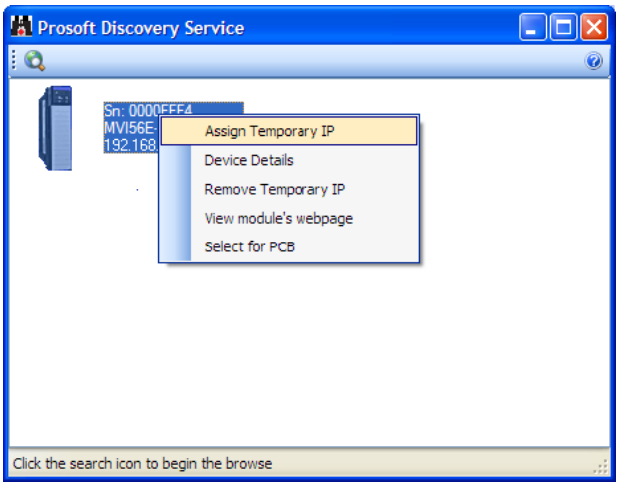


2 Click the right mouse button to open a shortcut menu. On the shortcut menu, choose **DIAGNOSTICS.**

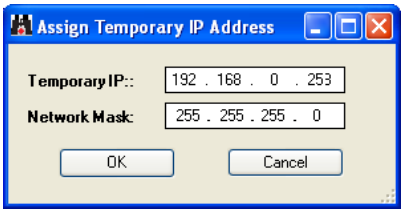


3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.

4 In the *Connection Setup* dialog box, click the **BROWSE DEVICE(S)** button to open the *ProSoft Discovery Service*. Right-click the module icon, and then choose **ASSIGN TEMPORARY IP**.

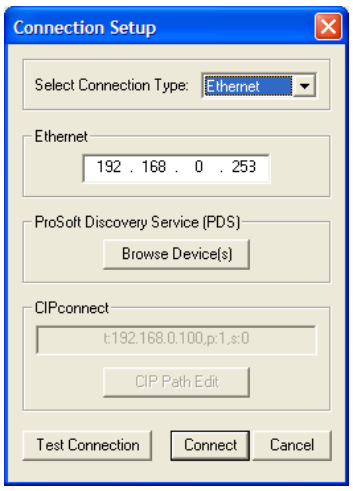


5 The module's default IP address is usually 192.168.0.250. Choose an unused IP within your subnet, and then click **OK**.

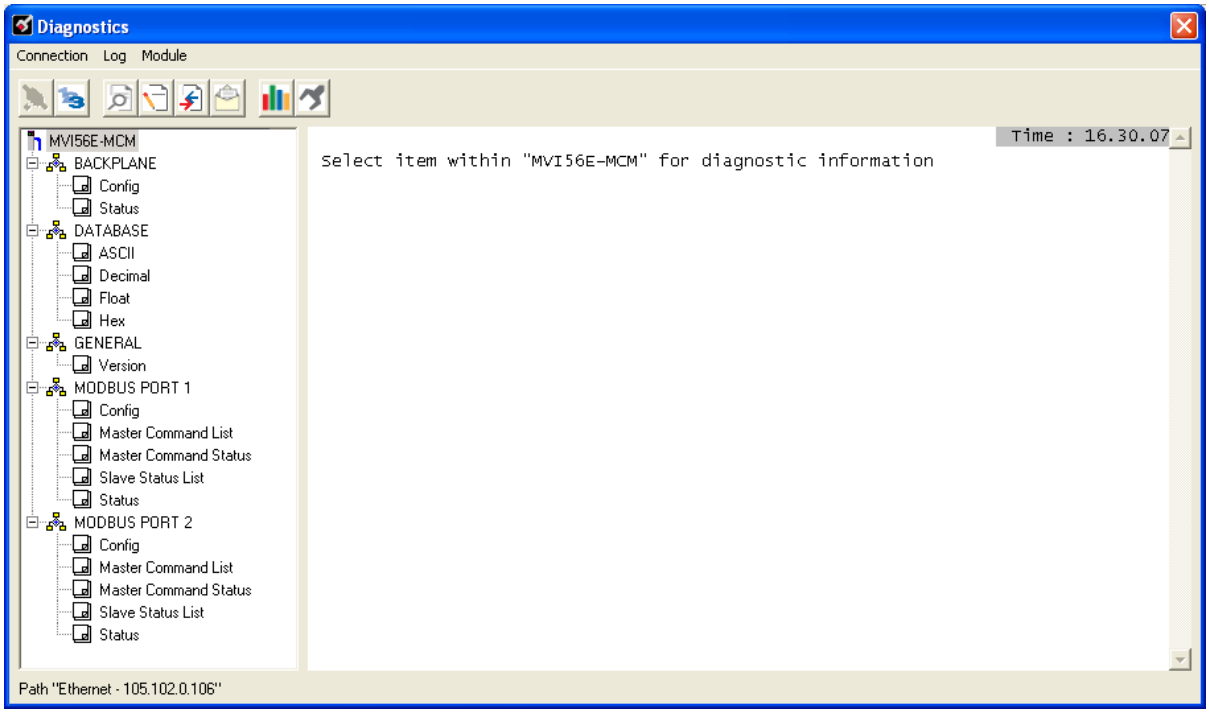


Important: The temporary IP address is only valid until the next time the module is initialized. For information on how to set the module's permanent IP address, see Assigning an IP Address in the Project (page 81).

6 Close the *ProSoft Discovery Service* window. Enter the temporary IP in the Ethernet address field of the *Connection Setup* dialog box, then click the **TEST CONNECTION** button to verify that the module is accessible with the current settings.



7 If the *Test Connection* is successful, click **CONNECT**. The *Diagnostics* menu displays in the *Diagnostics* window.



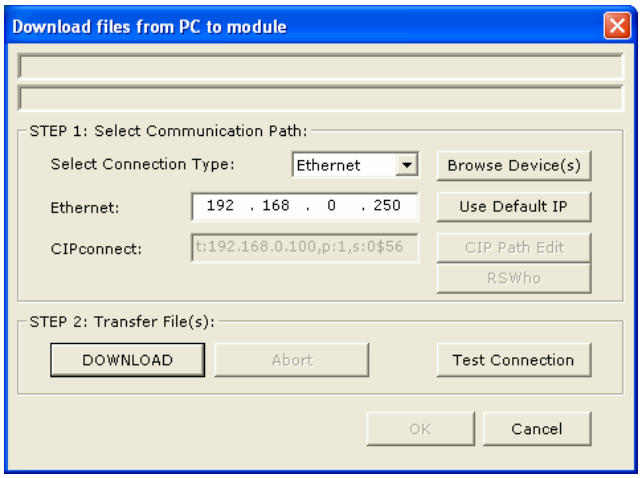
5.6 Downloading the Project to the Module

Note: For alternative methods of connecting to the module with your PC, refer to Connecting Your PC to the Module (page 83).

In order for the module to use the settings you configured, you must download (copy) the updated Project file from your PC to the module.

1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MCM** icon to open a shortcut menu.
2. Choose **DOWNLOAD FROM PC TO DEVICE.** This opens the *Download* dialog box.
3. In the *Download* dialog box, choose the connection type in the *Select Connection Type* dropdown box:
   - Choose **ETHERNET** if you are connecting to the module through the Ethernet cable.
   - Choose **1756 ENBT** if you are connecting to the module through CIPconnect or RSWho.
     Refer to Connecting Your PC to the Module (page 83) for more information.

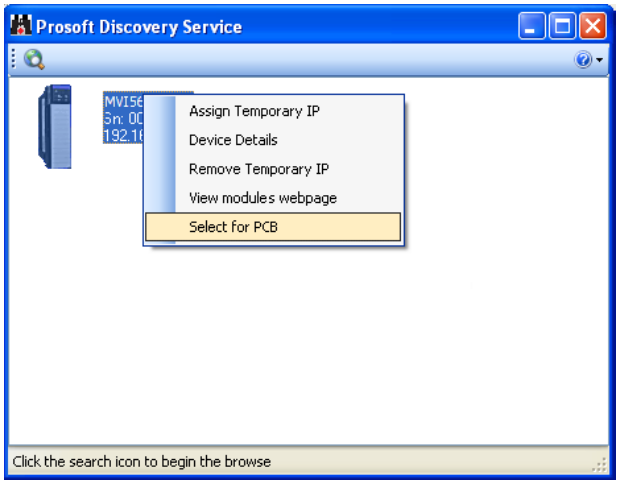
   Note: If you connected to the module using an Ethernet cable and set a temporary IP address, the Ethernet address field contains that temporary IP address. *ProSoft Configuration Builder* uses this temporary IP address to connect to the module.



4. Click **TEST CONNECTION** to verify that the IP address allows access to the module.
5. If the connection succeeds, click **DOWNLOAD** to transfer the Ethernet configuration to the module.

If the *Test Connection* procedure fails, you will see an error message. To correct the error, follow these steps.

1. Click **OK** to dismiss the error message.
2. In the *Download* dialog box, click **BROWSE DEVICE(S)** to open *ProSoft Discovery Service*.



3. Select the module, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **SELECT FOR PCB**.
4. Close *ProSoft Discovery Service*.
5. Click **DOWNLOAD** to transfer the configuration to the module.

5.7 Using the Diagnostics Menu in ProSoft Configuration Builder

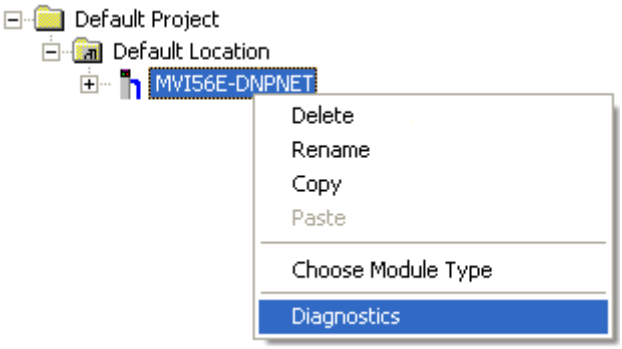
The *Diagnostics* menu, available through the Ethernet configuration port for this module, is arranged as a tree structure, with the *Main* menu at the top of the tree, and one or more submenus for each menu command. The first menu you see when you connect to the module is the *Main* menu.

Tip: You can have a ProSoft Configuration Builder *Diagnostics* window open for more than one module at a time.

To connect to the module's Configuration/Debug Ethernet port:

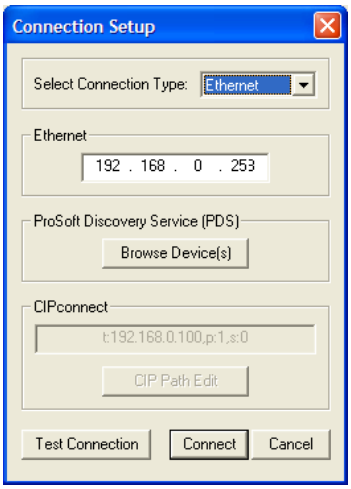
1. In *ProSoft Configuration Builder*, select the module, and then click the right mouse button to open a shortcut menu.

2 On the shortcut menu, choose **DIAGNOSTICS.**

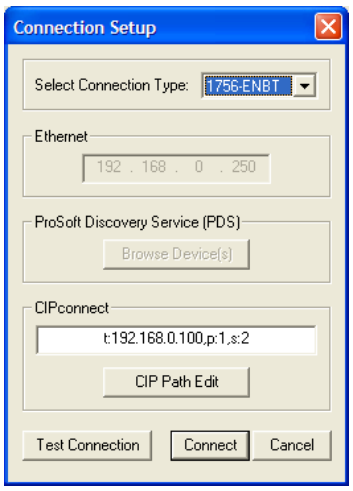


3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.

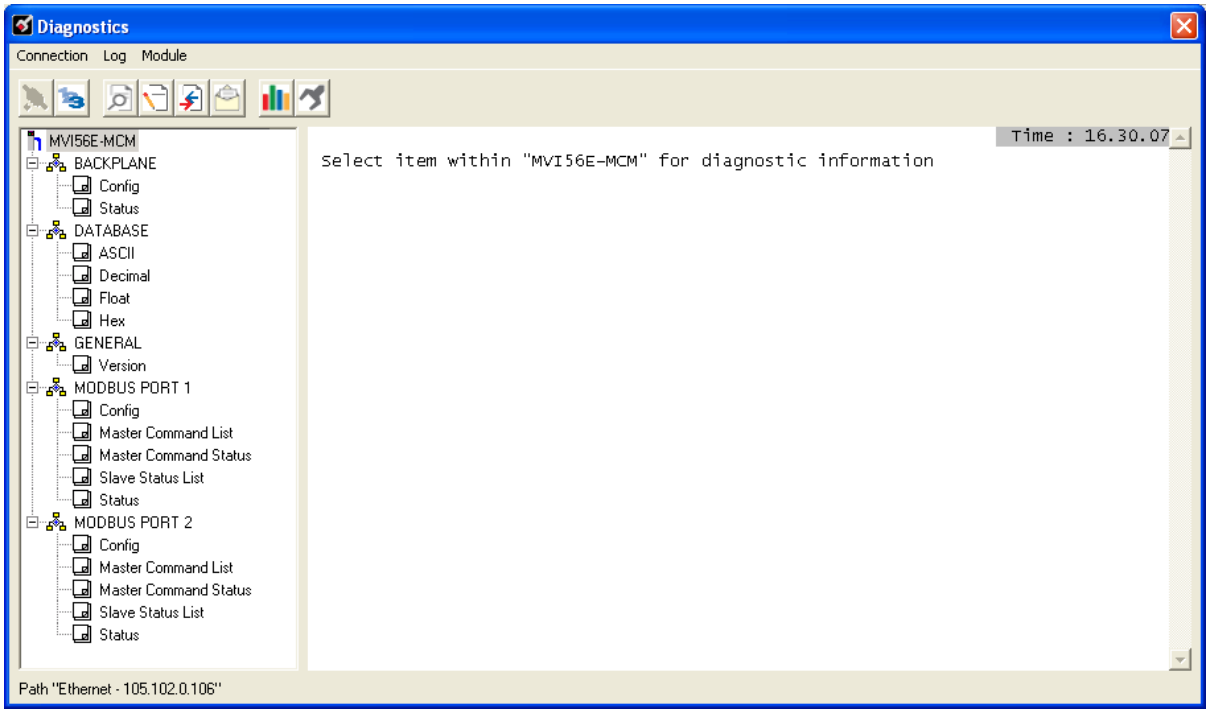
4 In the *Connection Setup* dialog box, click the **TEST CONNECTION** button to verify that the module is accessible with the current settings.



You can also use CIPconnect® to connect to the module through a 1756-ENBT card by choosing *1756-ENBT* in the **SELECT CONNECTION TYPE** list Refer to Using CIPconnect to Connect to the Module for information on how to construct a CIP path.

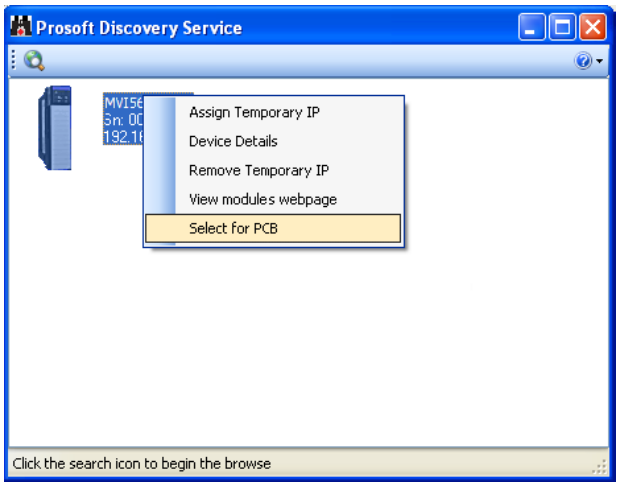


5 If the *Test Connection* is successful, click **CONNECT** to display the *Diagnostics* menu in the Diagnostics Window.



If *PCB* is unable to connect to the module:

1 Click the **BROWSE DEVICE(S)** button to open the *ProSoft Discovery Service*. Select the module, then right-click and choose **SELECT FOR PCB**.



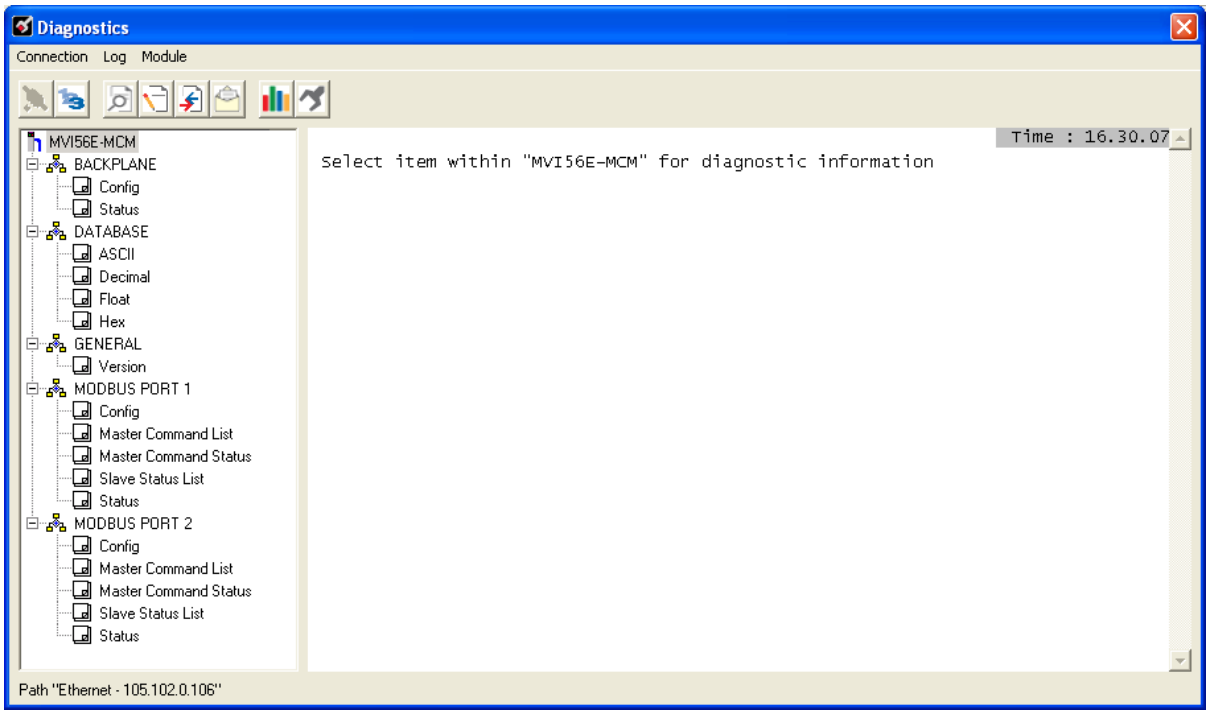
2 Close *ProSoft Discovery Service*, and click the **CONNECT** button again.

3 If these troubleshooting steps fail, verify that the Ethernet cable is connected properly between your computer and the module, either through a hub or switch (using the grey cable) or directly between your computer and the module (using the red cable).

If you are still not able to establish a connection, contact ProSoft Technology for assistance.

5.7.1 The Diagnostics Menu

The *Diagnostics* menu, available through the Ethernet configuration port for this module, is arranged as a tree structure, with the *Main* menu at the top of the tree, and one or more submenus for each menu command. The first menu you see when you connect to the module is the *Main* menu.



5.7.2 Monitoring Backplane Information

Use the *BACKPLANE* menu to view the backplane status information for the MVI56E-MCM module.

Backplane Configuration

Click Config to view current backplane configuration settings, including

- Read Start
- Read Count
- Write Start
- Write Count
- Error Status Pointer

The settings on this menu correspond with the **MCM.CONFIG.MODDEF** controller tags in the ModDef Settings (page 58).

Backplane Status

Use the *Status* menu to view current backplane status, including

- Number of retries
- Backplane status
- Fail count
- Number of words read
- Number of words written
- Number of words parsed
- Error count
- Event count
- Command count

During normal operation, the read, write, and parsing values should increment continuously, while the error value should not increment.

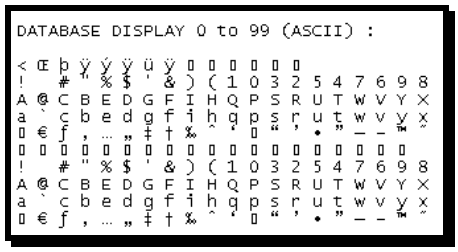
The status values on this menu correspond with members of the MVI56E-MCM Status Data Definition (page 152).

5.7.3 Monitoring Database Information

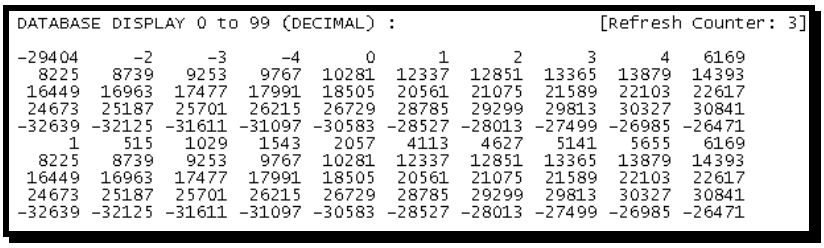
Use the *Database* menu to view the contents of the MVI56E-MCM module's internal database. The data locations on this menu corresponds with the MVI56E-MCM Database Definition (page 143)

You can view data in the following formats:

ASCII



Decimal



Float

```
DATABASE DISPLAY 0 to 49 (FLOAT) :                    [Refresh Counter: 5]

-1.#QNAN000E+000-1.#QNAN000E+000 9.18354962E-041 2.75509291E-040 1.97747944E-024
 2.21076282E-018 5.79887491E-016 6.44492959E-010 1.68752010E-007 4.41579286E-005
 4.88127480E+001 1.27530674E+004 1.40447017E+010 3.66483682E+012 9.55859400E+014
 1.04858893E+021 2.73179652E+023 2.98848446E+029 7.77852805E+031 2.02388230E+034
-1.93224776E-037-5.09760759E-035-5.74027813E-029-1.51029685E-026-3.97016299E-024
 9.62436112E-038 2.53936311E-035 2.86023979E-029 7.52614210E-027 1.97859390E-024
 2.21076282E-018 5.79887491E-016 6.44492959E-010 1.68752010E-007 4.41579286E-005
 4.88127480E+001 1.27530674E+004 1.40447017E+010 3.66483682E+012 9.55859400E+014
 1.04858893E+021 2.73179652E+023 2.98848446E+029 7.77852805E+031 2.02388230E+034
-1.93224776E-037-5.09760759E-035-5.74027813E-029-1.51029685E-026-3.97016299E-024
```

Hexadecimal

```
DATABASE DISPLAY 0 to 99 (HEXADECIMAL) :

907E FFFE FFFD FFFC 0000 0001 0002 0003 0004 1819
2021 2223 2425 2627 2829 3031 3233 3435 3637 3839
4041 4243 4445 4647 4849 5051 5253 5455 5657 5859
6061 6263 6465 6667 6869 7071 7273 7475 7677 7879
8081 8283 8485 8687 8889 9091 9293 9495 9697 9899
0001 0203 0405 0607 0809 1011 1213 1415 1617 1819
2021 2223 2425 2627 2829 3031 3233 3435 3637 3839
4041 4243 4445 4647 4849 5051 5253 5455 5657 5859
6061 6263 6465 6667 6869 7071 7273 7475 7677 7879
8081 8283 8485 8687 8889 9091 9293 9495 9697 9899
```

Use the scroll bar on the right edge of the window to view each page (100 words) of data.

5.7.4 Monitoring General Information

Use the General Menu to view module version information.

```
MVI56E-MCM > GENERAL > Version :

PRODUCT NAME CODE              :MCM5
SOFTWARE REVISION LEVEL        :2.01
OPERATING SYSTEM REVISION      :0109
RUN NUMBER                     :2801
PROGRAM SCAN COUNTER           :2473
BACKPLANE DRIVER VERSION       :1.2
BACKPLANE API VERSION          :1.0
MODULE NAME                    :MVI56E-MCM
VENDOR ID                      :309
DEVICE TYPE                    :12
PRODUCT CODE                   :5001
SERIAL NUMBER                  :0000FFF3
REVISION                       :2.1
```

The values on this menu correspond with the contents of the module's Misc. Status (page 150).

5.7.5 Monitoring Modbus Port Information

Use the Modbus Port 1 and Modbus Port 2 menus to view the information for each of the MVI56E-MCM module's Modbus application ports.

Port Configuration

Use the Port Configuration menu to view configuration settings for Modbus Port 1 and Modbus Port 2. The values on this menu correspond with the controller tags MCM.CONFIG.Port1 and MCMPort (page 155).

Master Command List

Use the Master Command List menu to view the command list settings for Modbus Port 1 and Modbus Port 2. The values on this menu correspond with the controller tags **MCM.CONFIG.PORT1MASTERCMD** and MCM.CONFIG.Port2MasterCmd.

Use the scroll bar on the right edge of the window to view each Modbus Master command.

Note: The Master Command List is available only if the port is configured as a Modbus Master.

Master Command Status

Use the Master Command Status menu to view Master command status for Modbus Port 1 and Modbus Port 2.

A zero indicates no error.

A non-zero value indicates an error. Refer to Command Error Codes (page 69) for an explanation of each value.

Slave Status List

Use the Slave Status List menu to view the status of each Slave connected to the Modbus Master port.

Slaves attached to the Master Port can have one of the following states:

0	The Slave is inactive and not defined in the command list for the Master Port.
1	The Slave is actively being polled or controlled by the Master Port. This does not indicate that the Slave has responded to this message.
2	The Master Port has failed to communicate with the Slave device. Communications with the Slave is suspended for a user defined period based on the scanning of the command list.
3	Communications with the Slave has been disabled by the ladder logic. No communication will occur with the Slave until this state is cleared by the ladder logic.

Refer to Slave Status Blocks (3000 to 3003 or 3100 to 3103) (page 123) for more information.

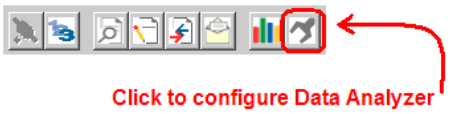
Port Status

Use the Port Status menu to view status for Modbus Port 1 and Modbus Port 2. During normal operation, the number of requests and responses should increment, while the number of errors should not change.

5.7.6 Data Analyzer

The Data Analyzer mode allows you to view all bytes of data transferred on each port. Both the transmitted and received data bytes are displayed. Use of this feature is limited without a thorough understanding of the protocol.

Configuring the Data Analyzer



Select Timing Interval

Time Ticks help you visualize how much data is transmitted on the port for a specified interval. Select the interval to display, or choose No Ticks to turn off timing marks.

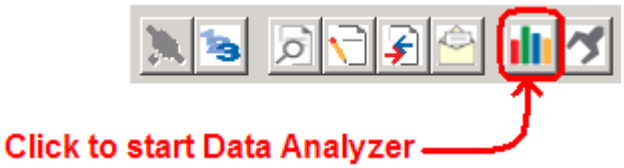
Select the Communication Port to Analyze

You can view incoming and outgoing data for one application port at a time. Choose the application port to analyze.

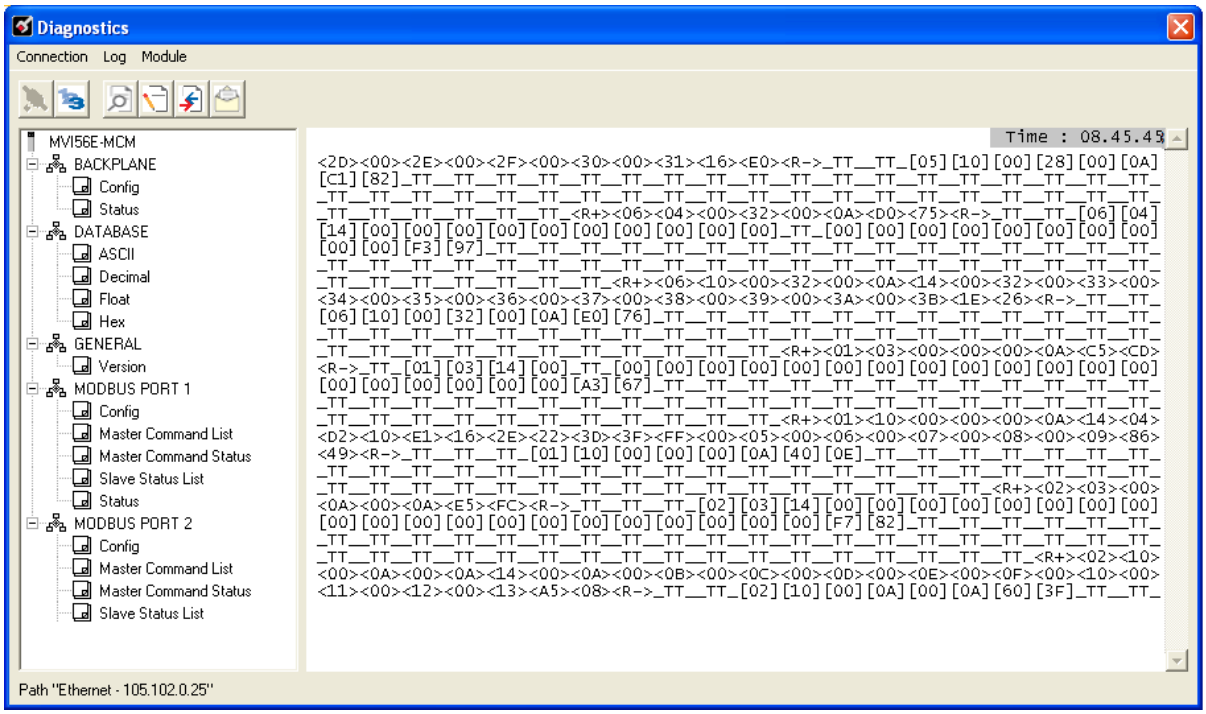
Select the Data Format

You can view incoming and outgoing data in Hexadecimal (HEX) or Alphanumeric (ASCII) format.

Starting the Data Analyzer



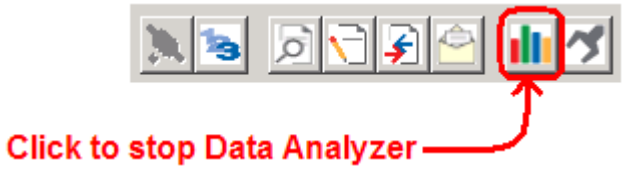
The following illustration shows an example of the Data Analyzer output.



The Data Analyzer can display the following special characters.

Character	Definition
[ ]	Data enclosed in these characters represent data received on the port.
< >	Data enclosed in these characters represent data transmitted on the port.
<R+>	These characters are inserted when the RTS line is driven high on the port.
<R->	These characters are inserted when the RTS line is dropped low on the port.
<CS>	These characters are displayed when the CTS line is recognized high.
_TT_	These characters are displayed when the "Time Tick" is set to any value other than "No Ticks".

Stopping the Data Analyzer



Important: When in analyzer mode, program execution will slow down. Only use this tool during a troubleshooting session. Before disconnecting from the Config/Debug port, please stop the data analyzer. This action will allow the module to resume its normal high speed operating mode.

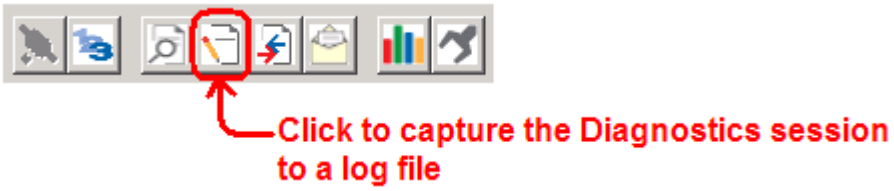
Data Analyzer Tips

For most applications, HEX is the best format to view the data, and this does include ASCII based messages (because some characters will not display in the Diagnostics window, and by capturing the data in HEX, we can figure out what the corresponding ASCII characters are supposed to be).

The Tick value is a timing mark. The module will print a _TT for every xx milliseconds of no data on the line. Usually 10milliseconds is the best value to start with.

To save a capture file of your Diagnostics session

1 After you have selected the Port, Format, and Tick, we are now ready to start a capture of this data.

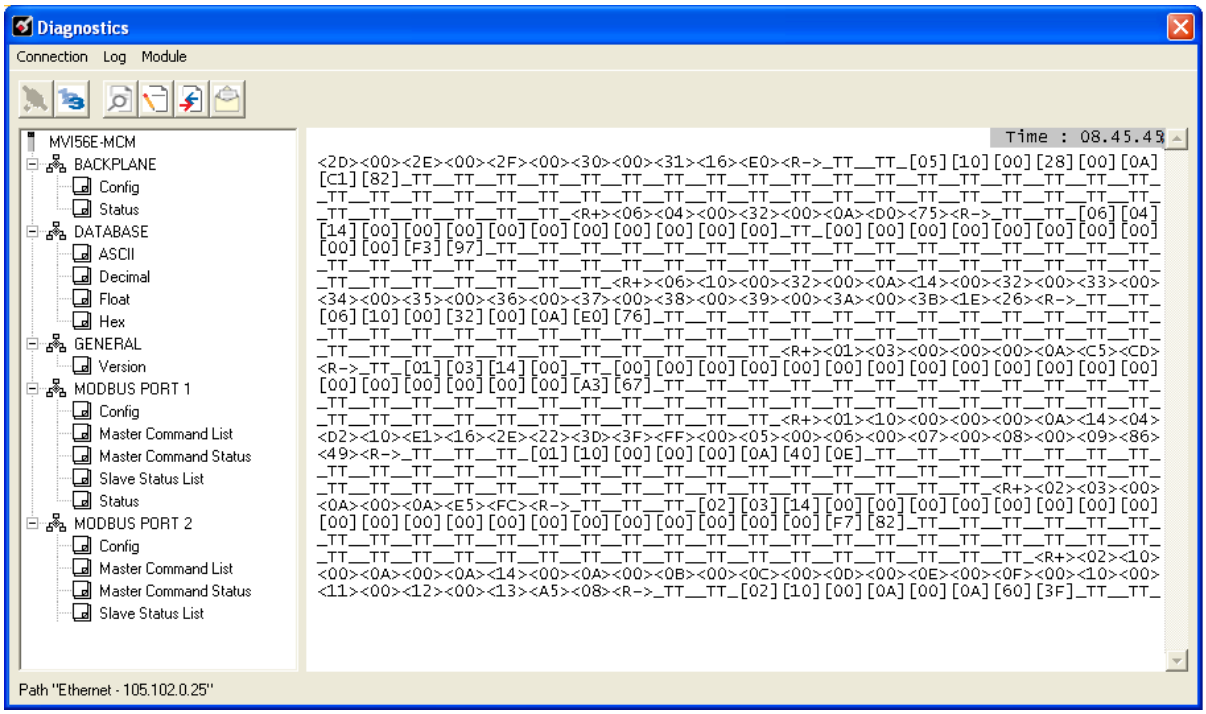


2 When you have captured the data you want to save, click again to stop capturing data.

You have now captured, and saved the file to your PC. This file can now be used in analyzing the communications traffic on the line, and assist in determining communication errors. The log file name is PCB-Log.txt, located in the root directory of your hard drive (normally Drive C).

Now you have everything that shows up on the Diagnostics screen being logged to a file called PCB-Log.txt. You can email this file to ProSoft Technical Support for help with issues on the communications network.

To begin the display of the communications data, start the Data Analyzer. When the Data Analyzer is running, you should see something like this.



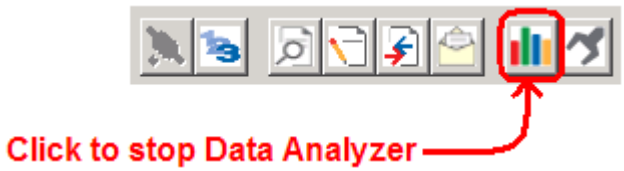
The <R+> means that the module is transitioning the communications line to a transmit state.

All characters shown in <> brackets are characters being sent out by the module.

The <R-> shows when the module is done transmitting data, and is now ready to receive information back.

And finally, all characters shown in the [ ] brackets is information being received from another device by the module.

After taking a minute or two of traffic capture, stop the Data Analyzer.



5.8 Reading Status Data from the Module

The MVI56E-MCM module returns a 33-word Status Data block that can be used to determine the module's operating status. This data is located in the module's database at registers 15270 to 15302 and at the location specified in the configuration. This data is transferred to the ControlLogix processor continuously with each read block. For a complete listing of the status data object, refer to MVI56E-MCM Status Data Definition (page 152).

5.8.1 Viewing the Error Status Table

Command execution status and error codes for each individual command are stored in a Master Command Status/Error List, held in the module's internal memory. There are several ways to view this data.

- View Command Status, Slave Status and Port Status in the Monitoring Modbus Port Information (page 105).
- Configure the Command Error Pointer parameter (<CmdErrPtr>) to copy the status/error values into the User Database area of module memory.
- Copy this table to a section of the ReadData area, where you can view it in the **<READDATAARRAY>** tag array in the ControlLogix controller tag database. You can use these values for communications status monitoring and alarming.
  - **<CMDERRPTR> = "MCM.CONFIG.PORTX.CMDERRPTR"**
  - **<READDATAARRAY> = "MCM.DATA.READDATA[X]"**

  These variables would hold the literal tag names in the sample program or Add-On Instruction. Use these variables to accommodate future ladder or tag changes while maintaining backward compatibility.

5.9 Configuration Error Codes

During module configuration download, the OK and APP LEDs will cycle through various states. If the OK LED remains RED and the APP LED remains OFF or RED for a long period of time, look at the configuration error words in the configuration request block. The structure of the block is shown in the following table.

Offset	Description	Length
0	Reserved	1
1	9000	1
2	Module Configuration Errors	1
3	Port 1 Configuration Errors	1
4	Port 2 Configuration Errors	1
5 to 248	Spare	244
249	-2 or -3	1

The bits in each configuration word are shown in the following table. The module configuration error word has the following definition:

Bit	Description	Value
0	Read block start value is greater than the database size.	0x0001
1	Read block start value is less than zero.	0x0002
2	Read block count value is less than zero.	0x0004
3	Read block count + start is greater than the database size.	0x0008
4	Write block start value is greater than the database size.	0x0010
5	Write block start value is less than zero.	0x0020
6	Write block count value is less than zero.	0x0040
7	Write block count + start is greater than the database size.	0x0080
8		0x0100
9		0x0200
10		0x0400
11		0x0800
12		0x1000
13		0x2000
14		0x4000
15		0x8000

The port configuration error words have the following definitions:

Bit	Description	Value
0	Type code is not valid. Enter a value from 0 (Master) to 1 (Slave).	0x0001
1	The float flag parameter is not valid.	0x0002
2	The float start parameter is not valid.	0x0004
3	The float offset parameter is not valid.	0x0008
4	Protocol parameter is not valid.	0x0010
5	Baud rate parameter is not valid.	0x0020
6	Parity parameter is not valid.	0x0040
7	Data bits parameter is not valid.	0x0080
8	Stop bits parameter is not valid.	0x0100
9	Slave ID is not valid.	0x0200
10	Input bit or word, output word and/or holding register offset(s) are not valid.	0x0400
11	Command count parameter is not valid.	0x0800
12	Spare	0x1000
13	Spare	0x2000
14	Spare	0x4000
15	Spare	0x8000

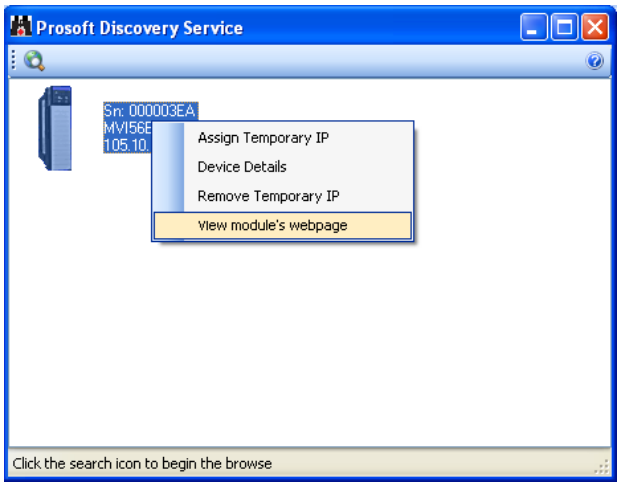
Correct any invalid data in the configuration for proper module operation. When the configuration contains a valid parameter set, all the bits in the configuration words will be clear. This does not indicate that the configuration is valid for the user application. Make sure each parameter is set correctly for the specific application.

Note: If the APP, BP ACT and OK LEDs blink at a rate of every one-second, this indicates a serious problem with the module. Call ProSoft Technology Support to arrange for repairs.

5.10 Connect to the Module's Web Page

The module's internal web server provides access to module status, diagnostics, and firmware updates.

1 In *ProSoft Discovery Service*, select the module to configure, and then click the right mouse button to open a shortcut menu.



2 On the shortcut menu, choose **VIEW MODULE'S WEBPAGE**.

6 Reference

In This Chapter

- Product Specifications 115
- Functional Overview 118
- Cable Connections 138
- MVI56E-MCM Database Definition 143
- MVI56E-MCM Configuration Data 144
- MVI56E-MCM Status Data Definition 152
- MVI56E-MCM User Defined Data Types 154
- Modbus Protocol Specification 160
- Using the Optional Add-On Instruction 172
- Using the Sample Program - RSLogix 5000 Version 15 and earlier 180

6.1 Product Specifications

The MVI56E Enhanced Modbus Master/Slave Communication Modules allow Rockwell Automation® ControlLogix® processors to easily interface with devices using the Modbus RTU/ASCII serial communications protocol.

The MVI56E-MCM and MVI56E-MCMXT act as input/output modules on the ControlLogix backplane, making Modbus data appear as I/O data to the processor. Data transfer to and from the processor is asynchronous from the communications on the Modbus network. Two independently configurable serial ports can operate on the same or different Modbus networks. Each port can be configured as a Modbus Master or Slave, sharing the same user-controlled, 10,000-word database.

The two modules are functionally the same. The MVI56E-MCM is designed for standard process applications. The MVI56E-MCMXT is designed for the Logix-XT™ control platform, allowing it to operate in extreme environments. It can tolerate higher operating temperatures, and it also has a conformal coating to protect it from harsh or caustic conditions.

6.1.1 General Specifications

- Backward-compatible with previous MVI56-MCM version
- Single Slot - 1756 ControlLogix® backplane compatible
- 10/100 MB Ethernet port for network configuration and diagnostics with Auto Cable Crossover Detection
- User-definable module data memory mapping of up to 10,000 16-bit registers

- CIPconnect®-enabled network diagnostics and monitoring using ControlLogix 1756-ENxT modules and EtherNet/IP® pass-thru communications
- Sample Ladder Logic or Add-On Instruction (AOI) used for data transfers between module and processor and for module configuration
- 4-character, scrolling, alphanumeric LED display of status and diagnostic data in plain English
- ProSoft Discovery Service (PDS) software finds the module on the network and assigns a temporary IP address to facilitate module access

6.1.2 General Specifications - Modbus Master/Slave

Communication Parameters	Baud rate: 110 to 115K baud Stop bits: 1 or 2 Data size: 7 or 8 bits Parity: None, Even, Odd RTS timing delays: 0 to 65535 milliseconds	
Modbus Modes	RTU mode (binary) with CRC-16 ASCII mode with LRC error checking	
Floating-Point Data	Floating-point data movement supported, including configurable support for Enron, Daniel®, and other implementations	
Modbus Function Codes Supported	1: Read Coil Status 2: Read Input Status 3: Read Holding Registers 4: Read Input Registers 5: Force (Write) Single Coil 6: Preset (Write) Single Holding Register 8: Diagnostics (Slave Only, Responds to Subfunction 00)	15: Force(Write) Multiple Coils 16: Preset (Write) Multiple Holding Registers 17: Report Slave ID (Slave Only) 22: Mask Write Holding Register (Slave Only) 23: Read/Write Holding Registers (Slave Only)

6.1.3 Functional Specifications

The MVI56E-MCM will operate on a Local or Remote rack (For remote rack applications with smaller data packet size please refer to the MVI56E-MCMR product)

- CIPconnect® enabled for module and network configuration using 1756-ENxT module with EtherNet/IP pass-through communications
- Supports Enron version of Modbus protocol for floating-point data transactions
- 4-digit LED Display for English based status and diagnostics information
- PCB includes powerful Modbus network analyzer
- Error codes, network error counters, and port status data available in user data memory

Slave Specifications

The MVI56E-MCM module accepts Modbus function code commands of 1, 2, 3, 4, 5, 6, 8, 15, 16, 17, 22, and 23 from an attached Modbus Master unit. A port configured as a Modbus Slave permits a remote Master to interact with all data contained in the module. This data can be derived from other Modbus Slave devices on the network, through a Master port, or from the ControlLogix processor.

Master Specifications

A port configured as a virtual Modbus Master device on the MVI56E-MCM module actively issues Modbus commands to other nodes on the Modbus network. 325 commands are supported on each port. Additionally, the Master ports have an optimized polling characteristic that polls slaves with communication problems less frequently. The ControlLogix processor ladder logic can issue commands directly from ladder logic or actively select commands from the command list to execute under ladder logic control.

Note: To use up to 325 commands, your MVI56E-MCM module needs to have firmware version 3.01 or higher, and your MVI56E-MCM Add-On Instruction needs to be version 2.8 or higher. Earlier versions support up to 100 commands.

6.1.4 Hardware Specifications

General	
Specification	**Description**
Backplane Current Load	800 mA @ 5 VDC 3 mA @ 24 VDC
Operating Temperature	0°C to 60°C (32°F to 140°F)
Storage Temperature	-40°C to 85°C (-40°F to 185°F)
Extreme/Harsh Environment	MVI56E-MCMXT comes with conformal coating
Shock	30 g operational 50 g non-operational Vibration: 5 g from 10 to 150 Hz
Relative Humidity	5% to 95% (without condensation)
LED Indicators	Battery Status (ERR) Application Status (APP) Module Status (OK)
4-Character, Scrolling, Alpha-Numeric LED Display	Shows Module, Version, IP, Port Client/Server Setting, Port Status, and Error Information
Communication Ethernet port	
Ethernet Port	10/100 Base-T, RJ45 Connector, for CAT5 cable Link and Activity LED indicators Auto-crossover cable detection
Shipped with Unit	5 foot Ethernet Straight-Thru Cable (Gray)

6.2 Functional Overview

6.2.1 About the Modbus Protocol

Modbus is a widely-used protocol originally developed by Modicon in 1978. Since that time, the protocol has been adopted as a standard throughout the automation industry.

The original Modbus specification uses a serial connection to communicate commands and data between Master and Slave devices on a network. Later enhancements to the protocol allow communication over other types of networks.

Modbus is a Master/Slave protocol. The Master establishes a connection to the remote Slave. When the connection is established, the Master sends the Modbus commands to the Slave. The MVI56E-MCM module can work as a Master and as a Slave.

The MVI56E-MCM module also works as an input/output module between itself and the Rockwell Automation backplane and ControlLogix processor. The module uses an internal database to pass data and commands between the processor and Master and Slave devices on Modbus networks.

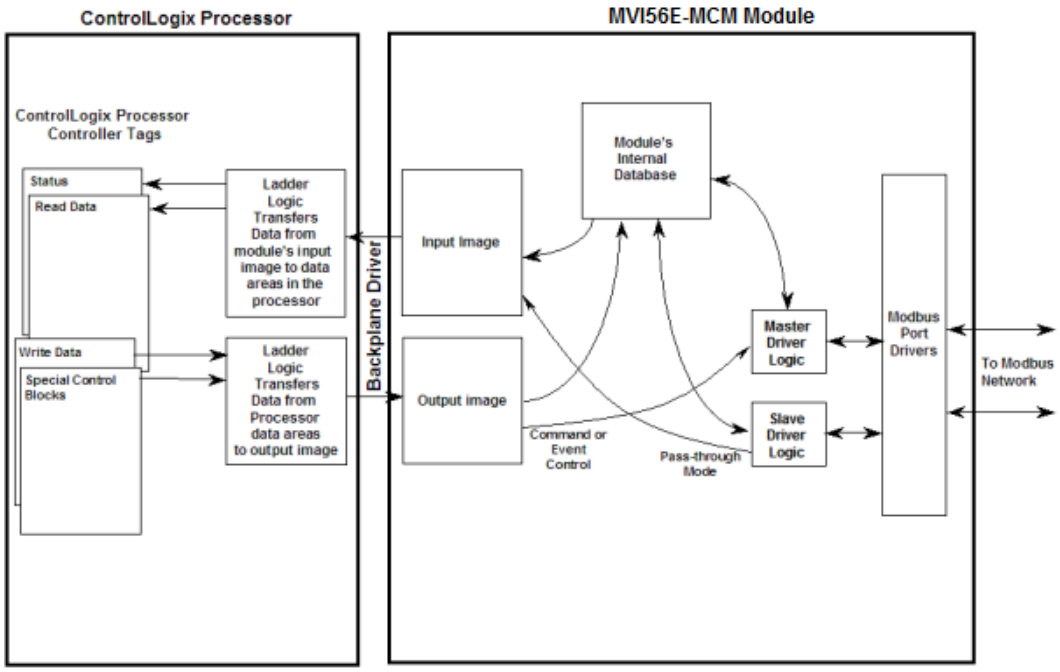
6.2.2 Backplane Data Transfer

The MVI56E-MCM module communicates directly over the ControlLogix backplane. Data is paged between the module and the ControlLogix processor across the backplane using the module's input and output images. The update frequency of the images is determined by the scheduled scan rate defined by the user for the module and the communication load on the module. Typical updates are in the range of 2 to 10 milliseconds.

This bi-directional transference of data is accomplished by the module filling in data in the module's input image to send to the processor. Data in the input image is placed in the Controller Tags in the processor by the ladder logic. The input image for the module is set to 250 words. This large data area permits fast throughput of data between the module and the processor.

The processor inserts data to the module's output image to transfer to the module. The module's program extracts the data and places it in the module's internal database. The output image for the module is set to 248 words. This large data area permits fast throughput of data from the processor to the module.

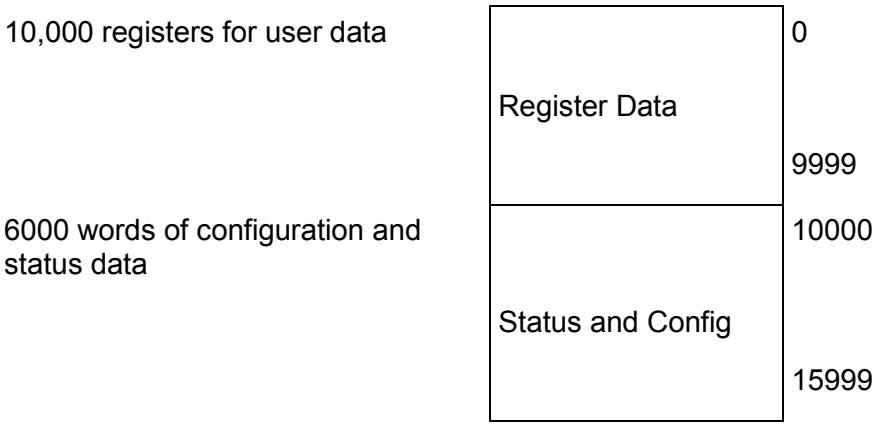
The following illustration shows the data transfer method used to move data between the ControlLogix processor, the MVI56E-MCM module and the Modbus Network.



As shown in the illustration above, all data transferred between the module and the processor over the backplane is through the input and output images. Ladder logic must be written in the ControlLogix processor to interface the input and output image data with data defined in the Controller Tags. All data used by the module is stored in its internal database. This database is defined as a virtual Modbus data table with addresses from 0 to 15999.

The database is translated into a Modbus data table, according to which a Modbus command is received or sent. Refer to Modbus Memory Map (page 59) for more information.

The following illustration shows the layout of the module's internal database structure:



Data contained in this database is paged through the input and output images by coordination of the ControlLogix ladder logic and the MVI56E-MCM module's program. Up to 248 words of data can be transferred from the module to the processor at a time. Up to 247 words of data can be transferred from the processor to the module. Each image has a defined structure depending on the data content and the function of the data transfer as defined below.

6.2.3 Normal Data Transfer

Normal data transfer includes the paging of the user data found in the module's internal database in registers 0 to 9999 and the status data. These data are transferred through read (input image) and write (output image) blocks. Refer to Using the Sample Program in an Existing Application (page 186) for a description of the data objects used with the blocks and the ladder logic required. The structure and function of each block is discussed below.

Read Block

These blocks of data transfer information from the module to the ControlLogix processor. The following table describes the structure of the input image.

Read Block from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID: -1 to 50	1
2 to 201	Read Data	200
202	Program Scan Counter	1
203 to 204	Product Code	2
205 to 206	Product Version	2
207 to 208	Operating System	2

Word Offset	Description	Length
209 to 210	Run Number	2
211 to 217	Port 1 Error Status	7
218 to 224	Port 2 Error Status	7
225 to 230	Data Transfer Status	6
231	Port 1 Current Error/Index	1
232	Port 1 Last Error/Index	1
233	Port 2 Current Error/Index	1
234	Port 2 Last Error/Index	1
235 to 248	Spare	14
249	Read Block ID	1

The Read Block ID is an index value used to determine the location of where the data will be placed in the ControlLogix processor controller tag array of module read data. Each transfer can move up to 200 words (block offsets 2 to 201) of data. In addition to moving user data, the block also contains status data for the module. This last set of data is transferred with each new block of data and is used for high-speed data movement.

The Write Block ID associated with the block requests data from the ControlLogix processor. Under normal program operation, the module sequentially sends read blocks and requests write blocks. For example, if the application uses three read and two write blocks, the sequence will be as follows:

R1W1→R2W2→R3W1→R1W2→R2W1→R3W2→R1W1→

This sequence will continue until interrupted by other write block numbers sent by the controller or by a command request from a node on the Modbus network or operator control through the module's Configuration/Debug port.

Write Block

These blocks of data transfer information from the ControlLogix processor to the module. The following table describes the structure of the output image.

Write Block from Processor to Module

Word Offset	Description	Length
0	Write Block ID: -1 to 50	1
1 to 200	Write Data	200
201 to 247	Spare	47

The Write Block ID is an index value used to determine the location in the module's database where the data will be placed. Each transfer can move up to 200 words (block offsets 1 to 200) of data.

6.2.4 Special Function Blocks

Special function blocks are optional blocks used to control the module or request special data from the module. The current version of the software supports the following special function blocks:

- Event Command
- Slave Status
- Command Control
- Module Configuration
- Master Command Data List
- Pass-Through
- Warm Boot
- Cold Boot
- Write Configuration

Event Command Blocks (1000 to 1255 or 2000 to 2255)

Event Command blocks send Modbus commands directly from the ladder logic to one of the Master Ports. The following table describes the format for these blocks.

Block Request from Processor to Module

Word Offset	Description	Length
0	1000 to 1255 or 2000 to 2255	1
1	Internal DB Address	1
2	Point Count	1
3	Swap Code	1
4	Modbus Function Code	1
5	Device Database Address	1
6 to 247	Spare	242

The block number defines the Modbus Port that will send the command, and the Slave node that will respond to the command. Blocks in the 1000 range are directed to Modbus Port 1, and blocks in the 2000 range are directed to Modbus Port 2. The Slave address is represented in the block number in the range of 0 to 255. The sum of these two values determines the block number. The other parameters passed with the block are used to construct the command.

- The **Internal DB Address** parameter specifies the module's database location to associate with the command
- The **Point Count** parameter defines the number of points or registers for the command
- The **Swap Code** is used with Modbus function 3 requests to change the word or byte order
- The **Modbus Function Code** has one of the following values 1, 2, 3, 4, 5, 6, 15, or 16
- The **Device Database Address** is the Modbus register or point in the remote Slave device to be associated with the command

When the command receives the block, it will process it and place it in the command queue. The module will respond to each command block with a read block. The following table describes the format of this block.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2	0 = Fail, 1 = Success	1
3 to 248	Spare	246
249	1000 to 1255 or 2000 to 2255	1

Word two of the block can be used by the ladder logic to determine if the command was added to the command queue of the module. The command will only fail if the command queue for the port is full (325 commands for each queue for modules with firmware version 3.01 or higher and Add-on-Instruction version 2.8 or higher).

Slave Status Blocks (3000 to 3003 or 3100 to 3103)

Slave status blocks send status information of each Slave device on a Master Port. Slaves attached to the Master Port can have one of the following states:

0	The Slave is inactive and not defined in the command list for the Master Port.
1	The Slave is actively being polled or controlled by the Master Port. This does not indicate that the Slave has responded to this message.
2	The Master Port has failed to communicate with the Slave device. Communications with the Slave is suspended for a user defined period based on the scanning of the command list.
3	Communications with the Slave has been disabled by the ladder logic. No communication will occur with the Slave until this state is cleared by the ladder logic.

Slaves are defined to the system when the module initializes the Master command list. Each Slave defined will be set to a state of one in this initial step. If the Master Port fails to communicate with a Slave device (retry count expired on a command), the Master will set the state of the Slave to a value of 2 in the status table. This suspends communication with the Slave device for a user specified scan count (**ERRORDELAYCNTR** value in the **MCMPORT** object for each port). Each time a command in the list is scanned that has the address of a suspended Slave, the delay counter value will be decremented. When the value reaches zero, the Slave state will be set to one. This will enable polling of the Slave.

Block ID	Description
3002	Request for first 128 Slave status values for Modbus Port 1
3003	Request for last 128 Slave status values for Modbus Port 1
3102	Request for first 128 Slave status values for Modbus Port 2
3103	Request for last 128 Slave status values for Modbus Port 2

The following table describes the format of these blocks.

Block Request from Processor to Module

Word Offset	Description	Length
0	3002 to 3003 or 3102 to 3103	1
1 to 247	Spare	246

The module will recognize the request by receiving the special write block code and respond with a read block with the following format:

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2 to 129	Slave Poll Status Data	128
130 to 248	Spare	119
249	3002 to 3003 or 3102 to 3103	1

Ladder logic can be written to override the value in the Slave status table. It can disable (state value of 3) by sending a special block of data from the processor to the Slave. Port 1 Slaves are disabled using block 3000, and Port 2 Slaves are disabled using block 3100. Each block contains the Slave node addresses to disable. The following table describes the structure of the block.

Block Request from Processor to Module

Word Offset	Description	Length
0	3000 or 3100	1
1	Number of Slaves in Block	1
2 to 201	Slave indexes	200
202 to 247	Spare	46

The module will respond with a block with the same identification code received and indicate the number of Slaves acted on with the block. The following table describes the format of the response block.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2	Number of Slaves processed	1
3 to 248	Spare	246
249	3000 or 3100	1

Ladder logic can be written to override the value in the Slave status table to enable the Slave (state value of 1) by sending a special block. Port 1 Slaves are enabled using block 3001, and Port 2 Slaves are enabled using block 3101. Each block contains the Slave node addresses to enable. The following table describes the format for this block.

Block Request from Processor to Module

Word Offset	Description	Length
0	3001 or 3101	1
1	Number of Slaves in Block	1
2 to 201	Slave indexes	200
202 to 247	Spare	46

The module will respond with a block with the same identification code received and indicate the number of Slaves acted on with the block. The following table describes the format of this response block.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2	Number of Slaves processed	1
3 to 248	Spare	246
249	3001 or 3101	1

Command Control Blocks (5001 to 5006 or 5101 to 5106)

Command Control blocks place commands in the command list into the command queue. Each port has a command queue of up to 325 commands (for modules with firmware version 3.01 or higher and Add-on-Instruction version 2.8 or higher). The module services commands in the queue before the Master command list. This gives high priority to commands in the queue. Commands placed in the queue through this mechanism must be defined in the Master command list. Under normal command list execution, the module will only execute commands with the Enable parameter set to one or two. If the value is set to zero, the command is skipped. Commands may be placed in the command list with an Enable parameter set to zero. These commands can then be executed using the Command Control blocks.

One to six commands can be placed in the command queue with a single request. The following table describes the format for this block.

Block Request from Processor to Module

Word Offset	Description	Length
0	5001 to 5006 or 5101 to 5106	1
1	Command index (**MCM.CONFIG.PORTXMASTERCMD** *[command index value]*)	1
2	Command index (**MCM.CONFIG.PORTXMASTERCMD** *[command index value]*)	1
3	Command index (**MCM.CONFIG.PORTXMASTERCMD** *[command index value]*)	1
4	Command index (**MCM.CONFIG.PORTXMASTERCMD** *[command index value]*)	1
5	Command index (**MCM.CONFIG.PORTXMASTERCMD** *[command index value]*)	1
6	Command index (**MCM.CONFIG.PORTXMASTERCMD** *[command index value]*)	1
7 to 247	Spare	241

Blocks in the range of 5001 to 5006 are used for Modbus Port 1, and blocks in the range of 5101 to 5106 are used for Modbus Port 2. The last digit in the block code defines the number of commands to process in the block. For example, a block code of 5003 contains 3 command indexes for Modbus Port 1. The Command index parameters in the block have a range of 0 to 99 and correspond to the Master command list entries.

The module responds to a Command Control block with a block containing the number of commands added to the command queue for the port. The following table describes the format for this block.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2	Number of commands added to command queue	1
3 to 248	Spare	246
249	5000 to 5006 or 5100 to 5106	1

Configuration Data Transfer

When the module performs a restart operation, it will request configuration information from the ControlLogix processor. This data is transferred to the module in specially formatted write blocks (output image). The module will poll for each block by setting the required write block number in a read block (input image). Refer to Using the Sample Program in an Existing Application (page 186) for a description of the data objects used with the blocks and the ladder logic required. The format of the blocks for configuration is given in the following topics.

Module Configuration Block (9000)

On boot-up, the module sends a request for configuration information to the processor. The request block has a Block ID of 9000.

Block Request from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	9000	1
2 to 248	Spare	247
249	9000	1

The processor responds by sending a block with general configuration information to the module.

Configuration Block from Processor to Module

Word Offset	Description	Length
0	9000	1
1 to 6	Backplane Setup	6
7 to 31	Port 1 Configuration	25
32 to 56	Port 2 Configuration	25
57 to 59	Port 1 Aux. Configuration	3
60 to 62	Port 2 Aux. Configuration	3
63 to 247	Spare	185

If the configuration information is valid, the module commences normal data transfer operation. If there are errors in the configuration, the module sends the processor a read block with configuration error codes.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	9000	1
2	Module Configuration Error Code	1
3	Port 1 Configuration Error Code	1
4	Port 2 Configuration Error Code	1
5 to 248	Spare	244
249	-2 or -3	1

Any errors must be corrected before the module will start operating.

Master Command Data List (6000 to 6012 or 6100 to 6112)

Each port on the module can be configured as a Modbus Master device containing its own list of 325 commands (for modules with firmware version 3.01 or higher and Add-on-Instruction version 2.8 or higher). The commands are read from the processor using the following Write Block IDs: Modbus Port 1: 6000 to 6012, and Modbus Port 2: 6100 to 6112. The module will sequentially poll for each block from the processor. Ladder logic must handle all of the data transfers. The following table describes the structure of each block.

Configuration Block from Processor to Module

Word Offset	Description	Length
0	6000 to 6012 and 6100 to 6112	1
1 to 8	Command Definition	8
9 to 16	Command Definition	8
17 to 24	Command Definition	8
25 to 32	Command Definition	8
33 to 40	Command Definition	8
41 to 48	Command Definition	8
49 to 56	Command Definition	8
57 to 64	Command Definition	8
65 to 72	Command Definition	8
73 to 80	Command Definition	8
81 to 88	Command Definition	8
89 to 96	Command Definition	8
97 to 104	Command Definition	8
105 to 112	Command Definition	8
113 to 120	Command Definition	8
121 to 128	Command Definition	8
129 to 136	Command Definition	8
137 to 144	Command Definition	8
145 to 152	Command Definition	8
153 to 160	Command Definition	8
161 to 168	Command Definition	8
169 to 176	Command Definition	8
177 to 184	Command Definition	8
185 to 192	Command Definition	8
193 to 200	Command Definition	8

Pass-Through Blocks

The Pass-through Mode allows a Modbus Slave port to pass write commands received from a host directly across the backplane to the ControlLogix processor for handling by ladder logic. Although this feature requires more ladder logic in order to implement a solution, there are certain situations where this functionality can be useful. Some of these situations include:

1. When the slave needs to know when it has been written to
2. When the acceptance of data may require some conditioning
3. When the host's write data registers must overlap the read register space

Unformatted Pass-Through Blocks (9996)

If one or more of the Slave Ports on the module are configured for the unformatted pass-through mode, the module will pass blocks with identification codes of 9996 to the processor for each received write command. Any Modbus function 5, 6, 15, and 16 commands will be passed from the port to the processor using this block identification number. Ladder logic must handle the receipt of all Modbus write functions to the processor and to respond as expected to commands issued by the remote Modbus Master device. The structure of the unformatted Pass-through block is shown in the following table.

Pass-Through Block 9996 from Module to Processor

Word Offset	Description	Length
0	0	1
1	9996	1
2	Number of bytes in Modbus message	1
3	Data address	1
4 to 248	Modbus message received	245
249	9996	1

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the Pass-through block with a write block with the following format.

Response Block 9996 from Processor to Module

Word Offset	Description	Length
0	9996	1
1 to 247	Spare	247

This will inform the module that the command has been processed and can be cleared from the pass-through queue.

Formatted Pass-Through Blocks (9956 to 9959)

If one or more of the Slave Ports on the module are configured for the Formatted Pass-through mode, the module will pass blocks with identification codes of 9956 to 9959 to the processor for each received write command. Any Modbus function 5, 6, 15 or 16 commands will be passed from the port to the processor using these block identification numbers. Ladder logic must handle the receipt of all Modbus write functions to the processor and must respond as expected to commands issued by the remote Modbus Master device. The structure of these formatted Pass-through blocks is shown in the following tables:

Function 5

Pass-Through Block 9958 from Module to Processor

Word Offset	Description	Length
0	0	1
1	9958	1
2	1	1
3	Bit Address	1
4 to 248	Modbus data received	245
249	9958	1

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the Pass-through block with a write block with the following format.

Response Block 9958 from Processor to Module

Word Offset	Description	Length
0	9958	1
1 to 247	Spare	247

This will inform the module that the command has been processed and can be cleared from the pass-through queue.

Function 6 and 16

Pass-Through Blocks 9956 or 9957 from Module to Processor

Offset	Description	Length
0	0	1
1	9956/9957 (Floating-point)	1
2	Number of data words	1
3	Data Address	1
4 to 248	Data	245
249	9956/9957	1

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the Pass-through block with a write block with the following format.

Response Blocks 9956 or 9957 from Processor to Module

Offset	Description	Length
0	9956/9957	1
1 to 247	Spare	247

This will inform the module that the command has been processed and can be cleared from the pass-through queue.

Function 15

When the module receives a function code 15 while in pass-through mode, the module will write the data using block ID 9959 for multiple-bit data. First the bit mask clears the bits to be updated. This is accomplished by ANDing the inverted mask with the existing data. Next the new data ANDed with the mask is ORed with the existing data. This protects the other bits in the INT registers from being affected.

Pass-Through Block 9959 from Module to Processor

Word Offset	Description	Length
0	0	1
1	9959	1
2	Number of Words	1
3	Word Address	1
4 to 53	Data	50
54 to 103	Mask	50
104 to 248	Spare	145
249	9959	1

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the Pass-through block with a write block with the following format.

Response Block 9959 from Processor to Module

Word Offset	Description	Length
0	9959	1
1 to 247	Spare	247

This will inform the module that the command has been processed and can be cleared from the pass-through queue.

Warm Boot Block (9998)

This block is sent from the ControlLogix processor to the module (output image) when the module is required to perform a warm-boot (software reset) operation. This block is commonly sent to the module any time configuration data modifications are made in the controller tags data area. This will cause the module to read the new configuration information and to restart. The following table describes the format of the Warm Boot block.

Block Request from Processor to Module

Word Offset	Description	Length
0	9998	1
1 to 247	Spare	247

Cold Boot Block (9999)

This block is sent from the ControlLogix processor to the module (output image) when the module is required to perform the cold boot (hardware reset) operation. This block is sent to the module when a hardware problem is detected by the ladder logic that requires a hardware reset. The following table describes the format of the Cold Boot block.

Block Request from Processor to Module

Word Offset	Description	Length
0	9999	1
1 to 247	Spare	247

MVI56E-MCM Remote Master Control

The MVI56E-MCM can receive special function block codes from a remote Master on the network to control the module, using specific values written to regions of this block. The module can respond to the following requests:

- Write configuration to processor
- Warm boot
- Cold boot

The remote Master controls the module by writing one of the following values to register 15400 (Modbus address 55401):

Block ID	Description
9997	Write configuration in database to the processor and warm boot the module.
9998	Warm boot the module.
9999	Cold boot the module.

The control register is reset to 0 after the operation is executed with the exception of the 9997 command. If the module fails to successfully transfer the configuration to the processor, it will place one of the following error codes in the control register.

Error Code	Description
0	No error, transfer successful
-1	Error transferring general configuration information.
-2	Error transferring Modbus Port 1 Master command list
-3	Error transferring Modbus Port 2 Master command list

Ladder logic must handle the 9997 command. No ladder logic is required for the warm or cold boot commands.

Write Configuration Block (-9000 and -6000 to -6003 or -6100 to -6103)

This special function is used to update the processor's module configuration information when the module's configuration has been altered by a remote Master. The remote Master writes a block code 9997 to module register 15400 (Modbus Address 55401), causing the module to write its current configuration to the processor. Ladder logic must handle the receipt of these blocks.

The first write block from the module contains a value of -9000 in the first word.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	-9000	1
2 to 7	Backplane Setup	6
8 to 32	Port 1 Configuration	25
33 to 57	Port 2 Configuration	25
58 to 60	Port 1 Configuration (continued)	3
61 to 63	Port 2 Configuration (continued)	3
64 to 248	Spare	185
249	-9000	1

Blocks -6000 to -6012 and -6100 to -6112 contain the Master Command List Data for ports 1 and 2, respectively:

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	-6000 to -6012 and -6100 to -6112	1
2 to 9	Command Definition	8
10 to 17	Command Definition	8
18 to 25	Command Definition	8
26 to 33	Command Definition	8
34 to 41	Command Definition	8
42 to 49	Command Definition	8

Word Offset	Description	Length
50 to 57	Command Definition	8
58 to 65	Command Definition	8
66 to 73	Command Definition	8
74 to 81	Command Definition	8
82 to 89	Command Definition	8
90 to 97	Command Definition	8
98 to 105	Command Definition	8
106 to 113	Command Definition	8
114 to 121	Command Definition	8
122 to 129	Command Definition	8
130 to 137	Command Definition	8
138 to 145	Command Definition	8
146 to 153	Command Definition	8
154 to 161	Command Definition	8
162 to 169	Command Definition	8
170 to 177	Command Definition	8
178 to 185	Command Definition	8
186 to 193	Command Definition	8
194 to 201	Command Definition	8
202 to 248	Spare	47
249	-6000 to -6012 and -6100 to -6112	1

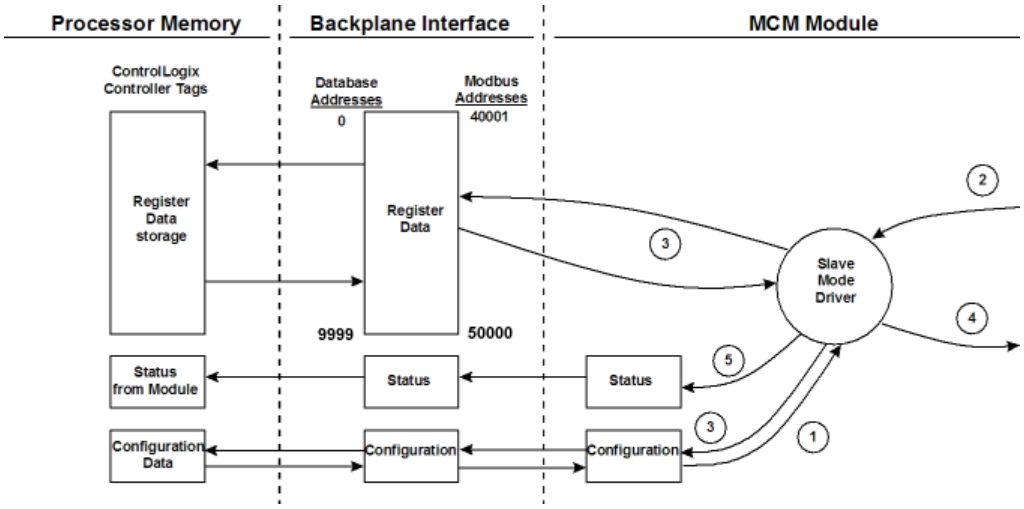
Each of these blocks must be handled by the ladder logic for proper module operation.

6.2.5 Data Flow Between MVI56E-MCM Module and ControlLogix Processor

The following topics describe the flow of data between the ControlLogix processor,MVI56E-MCM module, and nodes on the Modbus network. Each port on the module can be configured to emulate a Modbus Master device or a Modbus Slave device, independently from the configuration of the other port. Only the module database is shared between ports. The sections below discuss the operation of each mode.

Slave Driver

The Slave Driver Mode allows the module to respond to data read and write commands issued by a Master on the Modbus network. The following illustration describes the flow of data to and from the module.



1 The Modbus Slave Port driver receives the configuration information from the ControlLogix processor. This information configures the serial port and defines the Slave node characteristics. Additionally, the configuration information contains data that can be used to offset data in the database to addresses requested in messages received from Master units.

2 A Host device, such as a Modicon PLC or an HMI application, issues a read or write command to the module's node address. The port driver qualifies the message before accepting it into the module.

3 After the module accepts the command, the data is immediately transferred to or from the internal database in the module. If the command is a read command, the data is read from the database and a response message is built. If the command is a write command, the data is written directly into the database and a response message is built.

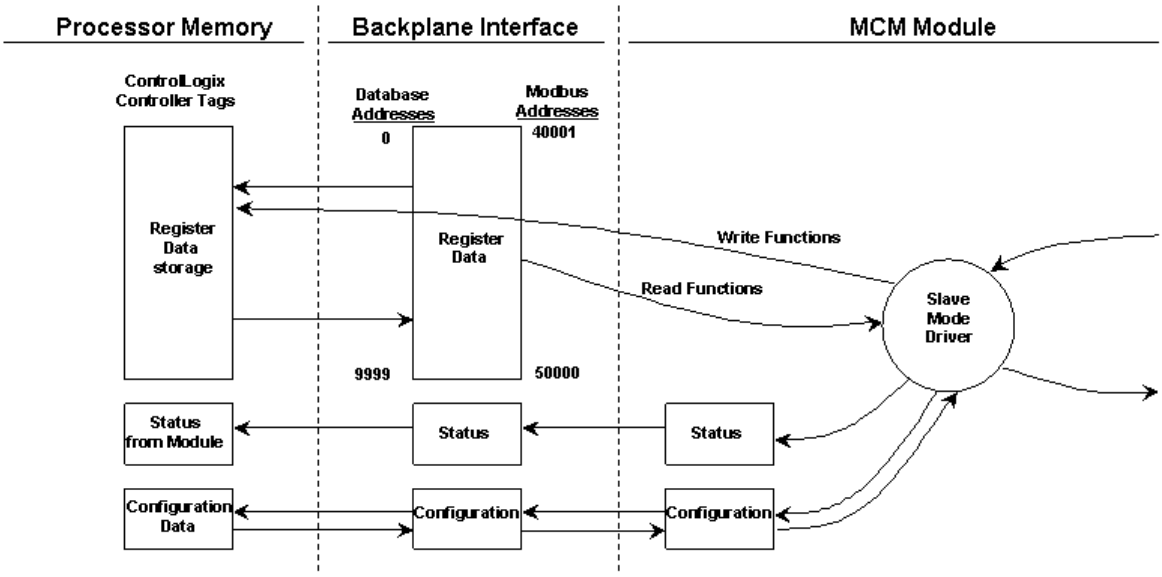
4 After the data processing has been completed in Step 2, the response is issued to the originating Master node.

5 Counters are available in the Status Block that permit the ladder logic program to determine the level of activity of the Slave Driver.

Refer to Using the Sample Program in an Existing Application (page 186) for a complete list of the parameters that must be defined for a Slave Port.

An exception to this normal mode is when the pass-through mode is implemented. In this mode, all write requests will be passed directly to the processor and will not be placed in the database. This permits direct, remote control of the processor without the intermediate database. This mode is especially useful for Master devices that do not send both states of control. For example, a SCADA system may only send an on command to a digital control point and never send the clear state. The SCADA system expects the local logic to reset the control bit. Pass-through must be used to simulate this mode.

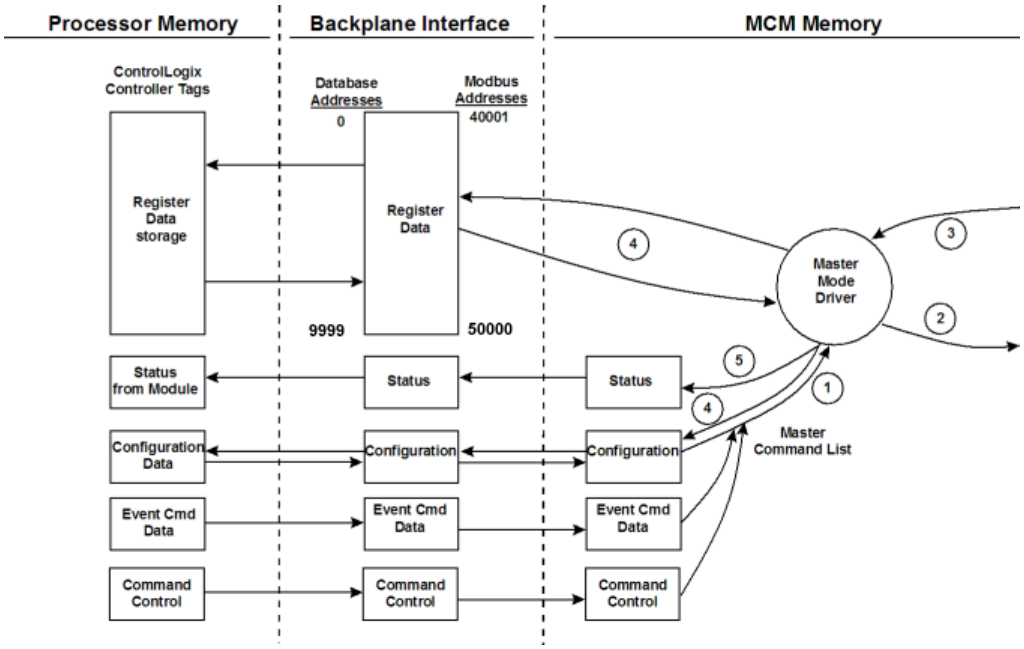
The following illustration shows the data flow for a Slave Port with pass-through enabled:



Master Driver

In the Master mode, the MVI56E-MCM module issues read or write commands to Slave devices on the Modbus network. These commands are user configured in the module via the Master Command List received from the ControlLogix processor or issued directly from the ControlLogix processor (event command control). Command status is returned to the processor for each individual command in the command list status block. The location of this status block in the module's internal database is user defined.

The following illustration describes the flow of data to and from the module.



1 The Master driver obtains configuration data from the ControlLogix processor. The configuration data obtained includes the number of commands and the Master Command List. These values are used by the Master driver to determine the type of commands to be issued to the other nodes on the Modbus network.

2 After configuration, the Master driver begins transmitting read and/or write commands to the other nodes on the network. If writing data to another node, the data for the write command is obtained from the module's internal database to build the command.

3 Presuming successful processing by the node specified in the command, a response message is received into the Master driver for processing.

4 Data received from the node on the network is passed into the module's internal database, assuming a read command.

5 Status is returned to the ControlLogix processor for each command in the Master Command List.

Refer to Using the Sample Program in an Existing Application (page 186) for a complete description of the parameters required to define the virtual Modbus Master Port.

Take care when constructing each command to ensure predictable operation of the module. If two commands write to the same internal database address of the module, the results will not be as desired. All commands containing invalid data are ignored by the module.

Master Command List

In order to function in the Master Mode, you must define the module's Master Command List. This list contains up to 325 individual entries (for module firmware versions 3.01 and higher and Add-on-Instruction version 2.8 or higher), with each entry containing the information required to construct a valid command. A valid command includes the following items:

- Command enable mode: (0) disabled, (1) continuous or (2) conditional
- Slave Node Address
- Command Type: Read or Write up to 125 words (16000 bits) per command
- Database Source and Destination Register Address: The addresses where data will be written or read.
- Count: The number of words to be transferred - 1 to 125 on FC 3, 4, or 16. Select the number of bits on FC 1, 2, 15.

As the list is read in from the processor and as the commands are processed, an error value is maintained in the module for each command. This error list can be transferred to the processor. The following tables describe the error codes generated by the module.

Note: 125 words is the maximum count allowed by the Modbus protocol. Some field devices may support less than the full 125 words. Check with your device manufacturer for the maximum count supported by your particular slave.

6.3 Cable Connections

The application ports on the MVI56E-MCM module support RS-232, RS-422, and RS-485 interfaces. Please inspect the module to ensure that the jumpers are set correctly to correspond with the type of interface you are using.

Note: When using RS-232 with radio modem applications, some radios or modems require hardware handshaking (control and monitoring of modem signal lines). Enable this in the configuration of the module by setting the UseCTS parameter to 1.

6.3.1 Ethernet Cable Specifications

The recommended cable is Category 5 or better. A Category 5 cable has four twisted pairs of wires, which are color-coded and cannot be swapped. The module uses only two of the four pairs.

The Ethernet ports on the module are Auto-Sensing. You can use either a standard Ethernet straight-through cable or a crossover cable when connecting the module to an Ethernet hub, a 10/100 Base-T Ethernet switch, or directly to a PC. The module will detect the cable type and use the appropriate pins to send and receive Ethernet signals.

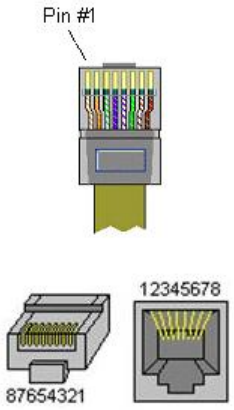
Ethernet cabling is like U.S. telephone cables, except that it has eight conductors. Some hubs have one input that can accept either a straight-through or crossover cable, depending on a switch position. In this case, you must ensure that the switch position and cable type agree.

Refer to Ethernet Cable Configuration (page 139) for a diagram of how to configure Ethernet cable.

6.3.2 Ethernet Cable Configuration

Note: The standard connector view shown is color-coded for a straight-through cable.

Crossover cable			Straight- through cable	
RJ-45 PIN	**RJ-45 PIN**		**RJ-45 PIN**	**RJ-45 PIN**
1 Rx+	3 Tx+		1 Rx+	1 Tx+
2 Rx-	6 Tx-		2 Rx-	2 Tx-
3 Tx+	1 Rx+		3 Tx+	3 Rx+
6 Tx-	2 Rx-		6 Tx-	6 Rx-



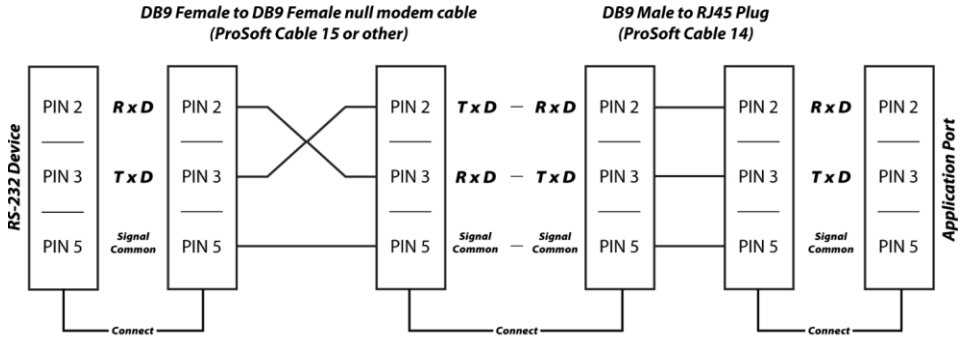
6.3.3 Ethernet Performance

High Ethernet traffic may impact MCM performance, consider one of these options:

- Use managed switches to reduce traffic coming to module port
- Use CIPconnect for these applications and disconnect the module Ethernet port from the network

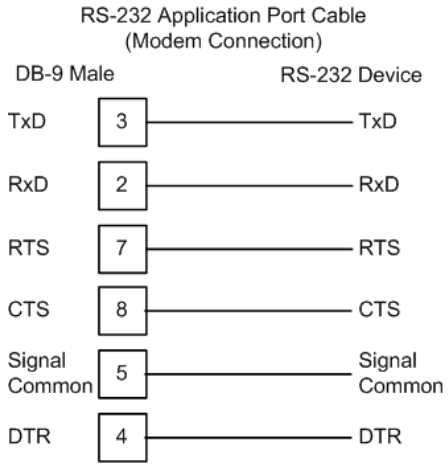
6.3.4 RS-232 Application Port(s)

When the RS-232 interface is selected, the use of hardware handshaking (control and monitoring of modem signal lines) is user definable. If no hardware handshaking will be used, here are the cable pinouts to connect to the port.



RS-232: Modem Connection (Hardware Handshaking Required)

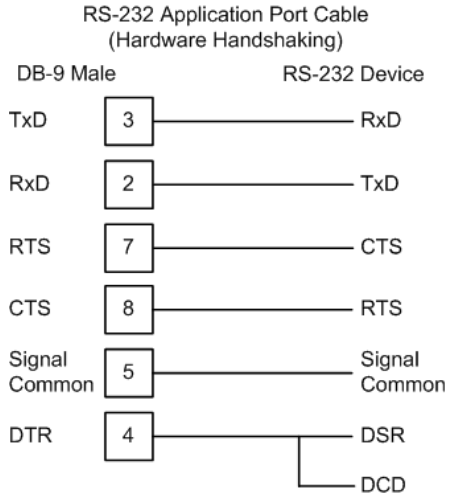
This type of connection is required between the module and a modem or other communication device.



The "Use CTS Line" parameter for the port configuration should be set to 'Y' for most modem applications.

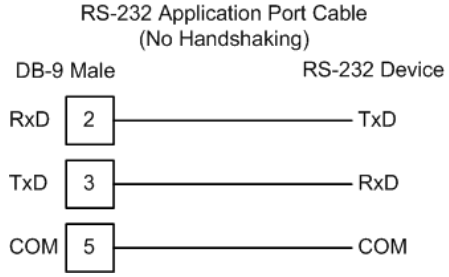
RS-232: Null Modem Connection (Hardware Handshaking)

This type of connection is used when the device connected to the module requires hardware handshaking (control and monitoring of modem signal lines).

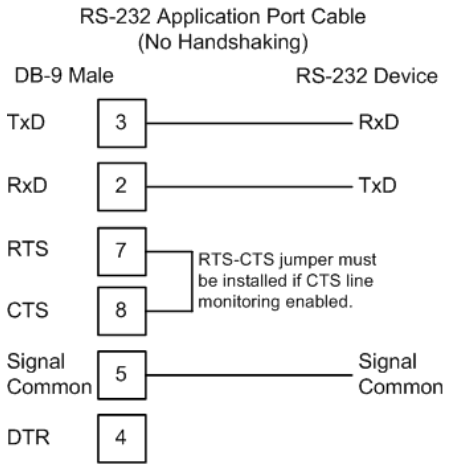


RS-232: Null Modem Connection (No Hardware Handshaking)

This type of connection can be used to connect the module to a computer or field device communication port.

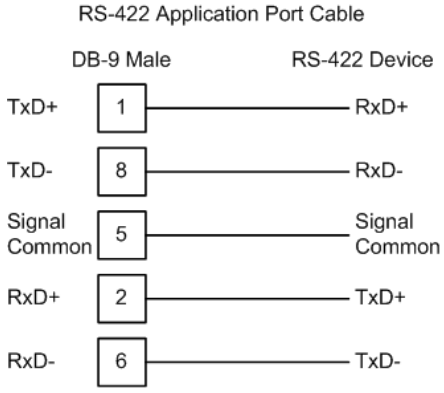


Note: For most null modem connections where hardware handshaking is not required, the *Use CTS Line* parameter should be set to **N** and no jumper will be required between Pins 7 (RTS) and 8 (CTS) on the connector. If the port is configured with the *Use CTS Line* set to **Y**, then a jumper is required between the RTS and the CTS lines on the port connection.



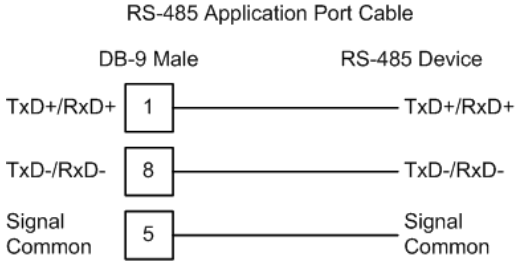
6.3.5 RS-422

The RS-422 interface requires a single four or five wire cable. The Common connection is optional, depending on the RS-422 network devices used. The cable required for this interface is shown below:



6.3.6 RS-485 Application Port(s)

The RS-485 interface requires a single two or three wire cable. The Common connection is optional, depending on the RS-485 network devices used. The cable required for this interface is shown below:

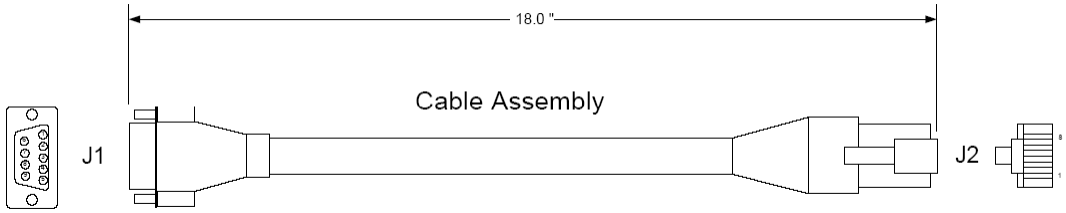


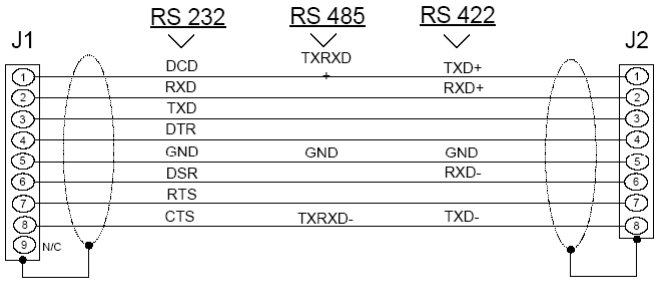
Note: Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In these cases, installing a 120-ohm terminating resistor between pins 1 and 8 on the module connector end of the RS-485 line may improve communication quality.

RS-485 and RS-422 Tip

If communication in the RS-422 or RS-485 mode does not work at first, despite all attempts, try switching termination polarities. Some manufacturers interpret + and -, or A and B, polarities differently.

6.3.7 DB9 to RJ45 Adaptor (Cable 14)





Wiring Diagram

6.4 MVI56E-MCM Database Definition

This section contains a listing of the internal database of the MVI56E-MCM module. This information can be used to interface other devices to the data contained in the module.

Register Range	Modbus Low	Modbus High	Content	Size
0 to 5999	40001	50000	User Data	10000
10000 to 10009	50001	50010	Backplane Configuration	10
10010 to 10039	50011	50040	Port 1 Setup	30
10040 to 10069	50041	50070	Port 2 Setup	30
10070 to 12669	50071	52670	Port 1 Commands	2600
12670 to 15269	52671	55270	Port 2 Commands	2600
15270 to 15359	55271	55350	Misc. Status Data	80

Register Range	Modbus Low	Modbus High	Content	Size
15350 to 15359	55351	55360	Port 1 Aux Setup	10
15360 to 15369	55361	55370	Port 2 Aux Setup	10
15400	55401		Command Control	1

The User Data area holds data collected from other nodes on the network (Master read commands) or data received from the processor (write blocks).

Additionally, this data area is used as a data source for the processor (read blocks) or other nodes on the network (write commands).

Detailed definition of the miscellaneous status data area can be found in MVI56E-MCM Status Data Definition (page 152).

Definition of the configuration data areas can be found in the data definition section of this document and in MVI56E-MCM Configuration Data (page 144).

Command Control (page 152) shows the Database register definition in a table for the Command Control block.

6.5 MVI56E-MCM Configuration Data

This section contains listings of the MVI56E-MCM module's database related to the module's configuration. This data is available to any node on the network and is read from the ControlLogix processor when the module first initializes.

6.5.1 Backplane Setup

Register	Content	Description
10,000	Write Start Reg	This parameter specifies the starting register in the module where the data transferred from the processor will be placed. Valid range for this parameter is 0 to 9999.
10,001	Write Reg Count	This parameter specifies the number of registers to transfer from the processor to the module. Valid entry for this parameter is 0 to 10000.
10,002	Read Start Reg	This parameter specifies the starting register in the module where data will be transferred from the module to the processor. Valid range for this parameter is 0 to 9999.
10,003	Read Reg Count	This parameter specifies the number of registers to be transferred from the module to the processor. Valid entry for this parameter is 0 to 10000.
10,004	Backplane Fail	This parameter specifies the number of successive transfer errors that must occur before the communication ports are shut down. If the parameter is set to zero, the communication ports will continue to operate under all conditions. If the value is set larger than 0 (1 to 65535), communications will cease if the specified number of failures occur.

Register	Content	Description
10,005	Error Status Pointer	This parameter specifies the register location in the module's database where module status data will be stored. If a value less than zero is entered, the data will not be stored in the database. If the value specified in the range of 0 to 9940, the data will be placed in the user data area.
10,006 to 10,009	Spare	

6.5.2 Port 1 Setup

Register	Content	Description
10,010	Enable	This parameter defines if this Modbus Port will be used. If the parameter is set to 0, the port is disabled. A value of 1 enables the port.
10,011	Type	This parameter specifies if the port will emulate a Modbus Master device (0), a Modbus Slave device without pass-through (1), a Modbus Slave device with unformatted pass-through (2), a Modbus Slave device with formatted pass-through and data swapping (3), or a Modbus Slave device with formatted pass-through and no data swapping (4).
10,012	Float Flag	This flag specifies if the floating-point data access functionality is to be implemented. If the float flag is set to 1, Modbus functions 3, 6, and 16 will interpret floating-point values for registers as specified by the two following parameters.
10,013	Float Start	This parameter defines the first register of floating-point data. All requests with register values greater than or equal to this value will be considered floating-point data requests. This parameter is only used if the Float Flag is enabled.
10,014	Float Offset	This parameter defines the start register for floating-point data in the internal database. This parameter is only used if the Float Flag is enabled.
10,015	Protocol	This parameter specifies the Modbus protocol to be used on the port. Valid protocols are: 0 = Modbus RTU and 1 = Modbus ASCII.
10,016	Baud Rate	This is the baud rate to be used on the port. Enter the baud rate as a value. For example, to select 19K baud, enter 19200. Valid entries are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 384 (for 38400 bps), 576 (for 57600 bps), and 115 (for 115,200 bps).
10,017	Parity	This is the parity code to be used for the port. Values are None, Odd, Even.
10,018	Data Bits	This parameter sets the number of data bits for each word used by the protocol. Valid entries for this field are 5 through 8.
10,019	Stop Bits	This parameter sets the number of stop bits for each data value sent. Valid entries are 1 and 2.

Register	Content	Description
10,020	RTS On	This parameter sets the number of milliseconds to delay after RTS is asserted before the data will be transmitted. Valid values are in the range of 0 to 65535 milliseconds.
10,021	RTS Off	This parameter sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal will be set low. Valid values are in the range of 0 to 65535.
10,022	Minimum Response Time	This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices.
10,023	Use CTS Line	This parameter specifies if the CTS modem control line is to be used. If the parameter is set to 0, the CTS line will not be monitored. If the parameter is set to 1, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire).
10,024	Slave ID	This parameter defines the virtual Modbus Slave address for the internal database. All requests received by the port with this address are processed by the module. Verify that each device has a unique address on a network. Valid range for this parameter is 1 to 255 (247 on some networks).
10,025	Bit in Offset	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus Function 2 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
10,026	Word in Offset	This parameter specifies the offset address in the internal Modbus database for network request for Modbus function 4 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
10,027	Out in Offset	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 1, 5, or 15 commands. For example, if the value is set to 100, an address request of 0 will correspond to register 100 in the database.
10,028	Holding Reg Offset	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 3, 6, or 16 commands. For example, if a value of 50 is entered, a request for address 0 will correspond to the register 50 in the database.
10,029	Command Count	This parameter specifies the number of commands to be processed by the Modbus Master Port.

Register	Content	Description
10,030	Minimum Command Delay	This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries.
10,031	Command Error Pointer	This parameter sets the address in the internal Modbus database where the command error will be placed. If the value is set to -1, the data will not be transferred to the database. The valid range of values for this parameter is -1 to 9675.
10,032	Response Timeout	This parameter represents the message response timeout period in 1-millisecond increments. This is the time that a port configured as a Master will wait before re-transmitting a command if no response is received from the addressed Slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network.
10,033	Retry Count	This parameter specifies the number of times a command will be retried if it fails. If the Master Port does not receive a response after the last retry, the Slave devices communication will be suspended on the port for Error Delay Counter scans.
10,034	Error Delay Counter	This parameter specifies the number of poll attempts to be skipped before trying to re-establish communications with a slave that has failed to respond to a command within the time limit set by the *Response Timeout* parameter. After the slave fails to respond, the master will skip sending commands that should have been sent to the slave until the number of skipped commands matches the value entered in this parameter. This creates a sort of *slow poll* mode for slaves that are experiencing communication problems.
10,035 to 10,039	Spare	Reserved for future use.

6.5.3 Port 2 Setup

Register	Content	Description
10,040	Enable	This parameter defines if this Modbus Port will be used. If the parameter is set to 0, the port is disabled. A value of 1 enables the port.
10,041	Type	This parameter specifies if the port will emulate a Modbus Master device (0), a Modbus Slave device without pass-through (1), a Modbus Slave device with unformatted pass-through (2), a Modbus Slave device with formatted pass-through and data swapping (3), or a Modbus Slave device with formatted pass-through and no data swapping (4).

Register	Content	Description
10,042	Float Flag	This flag specifies if the floating-point data access functionality is to be implemented. If the float flag is set to 1, Modbus functions 3, 6, and 16 will interpret floating-point values for registers as specified by the two following parameters.
10,043	Float Start	This parameter defines the first register of floating-point data. All requests with register values greater than or equal to this value will be considered floating-point data requests. This parameter is only used if the Float Flag is enabled.
10,044	Float Offset	This parameter defines the start register for floating-point data in the internal database. This parameter is only used if the Float Flag is enabled.
10,045	Protocol	This parameter specifies the Modbus protocol to be used on the port. Valid protocols are: 0 = Modbus RTU and 1 = Modbus ASCII.
10,046	Baud Rate	This is the baud rate to be used on the port. Enter the baud rate as a value. For example, to select 19K baud, enter 19200. Valid entries are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 384 (for 38400 bps), 576 (for 57600 bps), and 115 (for 115,200 bps).
10,047	Parity	This is the parity code to be used for the port. Values are None, Odd, Even.
10,048	Data Bits	This parameter sets the number of data bits for each word used by the protocol. Valid entries for this field are 5 through 8.
10,049	Stop Bits	This parameter sets the number of stop bits for each data value sent. Valid entries are 1 and 2.
10,050	RTS On	This parameter sets the number of milliseconds to delay after RTS is asserted before the data will be transmitted. Valid values are in the range of 0 to 65535 milliseconds.
10,051	RTS Off	This parameter sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal will be set low. Valid values are in the range of 0 to 65535.
10,052	Minimum Response Time	This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices.
10,053	Use CTS Line	This parameter specifies if the CTS modem control line is to be used. If the parameter is set to 0, the CTS line will not be monitored. If the parameter is set to 1, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire).

Register	Content	Description
10,054	Slave ID	This parameter defines the virtual Modbus Slave address for the internal database. All requests received by the port with this address are processed by the module. Verify that each device has a unique address on a network. Valid range for this parameter is 1 to 255 (247 on some networks).
10,055	Bit in Offset	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus Function 2 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
10,056	Word in Offset	This parameter specifies the offset address in the internal Modbus database for network request for Modbus function 4 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
10,057	Out in Offset	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 1, 5, or 15 commands. For example, if the value is set to 100, an address request of 0 will correspond to register 100 in the database.
10,058	Holding Reg Offset	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 3, 6, or 16 commands. For example, if a value of 50 is entered, a request for address 0 will correspond to the register 50 in the database.
10,059	Command Count	This parameter specifies the number of commands to be processed by the Modbus Master Port.
10,060	Minimum Command Delay	This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries.
10,061	Command Error Pointer	This parameter sets the address in the internal Modbus database where the command error will be placed. If the value is set to -1, the data will not be transferred to the database. The valid range of values for this parameter is -1 to 9675.
10,062	Response Timeout	This parameter represents the message response timeout period in 1-millisecond increments. This is the time that a port configured as a Master will wait before re-transmitting a command if no response is received from the addressed Slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network.
10,063	Retry Count	This parameter specifies the number of times a command will be retried if it fails. If the Master Port does not receive a response after the last retry, the Slave devices communication will be suspended on the port for Error Delay Counter scans.

Register	Content	Description
10,064	Error Delay Counter	This parameter specifies the number of poll attempts to be skipped before trying to re-establish communications with a slave that has failed to respond to a command within the time limit set by the *Response Timeout* parameter. After the slave fails to respond, the master will skip sending commands that should have been sent to the slave until the number of skipped commands matches the value entered in this parameter. This creates a sort of *slow poll* mode for slaves that are experiencing communication problems.
10,065 to 10,069	Spare	

6.5.4 Port 1 Commands

Register	Content	Description
10,070 to 10,077	Command #1	This set of registers contains the parameters for the first command in the Master command list. Refer to Master Command Configuration (page 37).
10,078 to 10,085	Command #2	Command #2 data set
-	-	-
12,662 to 12,669	Command #325	Command #325 data set

Note: To use up to 325 commands, your MVI56E-MCM module needs to have firmware version 3.01 or higher, and your MVI56E-MCM Add-On Instruction needs to be version 2.8 or higher. Earlier versions support up to 100 commands.

6.5.5 Port 2 Commands

Register	Content	Description
12,670 to 12,677	Command #1	This set of registers contains the parameters for the first command in the Master command list. Refer to Master Command Configuration (page 37).
12,678 to 12,685	Command #2	Command #2 data set
-	-	-
15,626 to 15,629	Command #325	Command #325 data set

6.5.6 Misc. Status

Register	Content	Description
15,270	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
15,271 to 15,272	Product Code	These two registers contain the product code of "MCM".

Register	Content	Description
15,273 to 15,274	Product Version	These two registers contain the product version for the current running software.
15,275 to 15,276	Operating System	These two registers contain the month and year values for the program operating system.
15,277 to 15,278	Run Number	These two registers contain the run number value for the currently running software.
15,279	Port 1 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
15,280	Port 1 Command List Response	This field contains the number of Slave response messages received on the port.
15,281	Port 1 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
15,282	Port 1 Requests	This field contains the total number of messages sent from the port.
15,283	Port 1 Responses	This field contains the total number of messages received on the port.
15,284	Port 1 Errors Sent	This field contains the total number of message errors sent from the port.
15,285	Port 1 Errors Received	This field contains the total number of message errors received on the port.
15,286	Port 2 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
15,287	Port 2 Command List Response	This field contains the number of Slave response messages received on the port.
15,288	Port 2 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
15,289	Port 2 Requests	This field contains the total number of messages sent out the port.
15,290	Port 2 Responses	This field contains the total number of messages received on the port.
15,291	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
15,292	Port 2 Errors Received	This field contains the total number of message errors received on the port.
15,293	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
15,294	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
15,295	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
15,296	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
15,297	Command Block Count	This field contains the total number of command blocks received from the processor.

Register	Content	Description
15,298	Error Block Count	This field contains the total number of block errors recognized by the module.
15,299	Port 1 Current Error	For a Slave Port, this field contains the value of the current error code returned. For a Master Port, this field contains the index of the currently executing command.
15,300	Port 1 Last Error	For a Slave Port, this field contains the value of the last error code returned. For a Master Port, this field contains the index of the command with the error.
15,301	Port 2 Current Error	For a Slave Port, this field contains the value of the current error code returned. For a Master Port, this field contains the index of the currently executing command.
15,302	Port 2 Last Error	For a Slave Port, this field contains the value of the last error code returned. For a Master Port, this field contains the index of the command with an error.
15,303 to 15,350	Spare	
15,351	Port 1 InterCharacterDelay	0 to 65535 milliseconds time between characters to signal end of message
15,352	Port 1 Fcn 99 Offset	Internal DB offset to Function 99 counter.
15,353 to 15,360	Spare	
15,360	Spare	
15,361	Port 2 InterCharacterDelay	0 to 65535 milliseconds time between characters to signal end of message
15,362	Port 2 Fcn 99 Offset	Internal DB offset to Function 99 counter.
15,363 to 15,399	Spare	

6.5.7 Command Control

Register	Content	Description
15,400	Command Code	Enter one of the valid control command codes in this register to control the module (9997, 9998, or 9999).

6.6 MVI56E-MCM Status Data Definition

This section contains a description of the members present in the **MCM.STATUS** object. This data is transferred from the module to the processor as part of each read block.

Offset	Content	Description
202	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
203 to 204	Product Code	These two registers contain the product code of "MCM".
205 to 206	Product Version	These two registers contain the product version for the current running software.

Offset	Content	Description
207 to 208	Operating System	These two registers contain the month and year values for the program operating system.
209 to 210	Run Number	These two registers contain the run number value for the currently running software.
211	Port 1 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
212	Port 1 Command List Response	This field contains the number of Slave response messages received on the port.
213	Port 1 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
214	Port 1 Requests	This field contains the total number of messages sent from the port.
215	Port 1 Responses	This field contains the total number of messages received on the port.
216	Port 1 Errors Sent	This field contains the total number of message errors sent from the port.
217	Port 1 Errors Received	This field contains the total number of message errors received on the port.
218	Port 2 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
219	Port 2 Command List Response	This field contains the number of Slave response messages received on the port.
220	Port 2 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
221	Port 2 Requests	This field contains the total number of messages sent out the port.
222	Port 2 Responses	This field contains the total number of messages received on the port.
223	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
224	Port 2 Errors Received	This field contains the total number of message errors received on the port.
225	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
226	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
227	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
228	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
229	Command Block Count	This field contains the total number of command blocks received from the processor.
230	Error Block Count	This field contains the total number of block errors recognized by the module.

Offset	Content	Description
231	Port 1 Current Error	For a Slave Port, this field contains the value of the most recently returned error code. For a Master Port, this field contains the index number of the most recently executed command that failed.
232	Port 1 Last Error	For a Slave Port, this field contains the value of the previous most recently returned error code. For a Master Port, this field contains the index number of the previous most recently executed command that failed.
233	Port 2 Current Error	For a Slave Port, this field contains the value of the most recently returned error code. For a Master Port, this field contains the index number of the most recently executed command that failed.
234	Port 2 Last Error	For a Slave Port, this field contains the value of the previous most recently returned error code. For a Master Port, this field contains the index number of the previous most recently executed command that failed.

6.7 MVI56E-MCM User Defined Data Types

6.7.1 MCMModuleDef

This object contains the data types that apply to the operation of the module.

Name	Data Type	Description
CONFIG	MCMCONFIG (page 154)	Module and port configuration
DATA	MCMDATA (page 157)	Modbus data transferred between module and processor
STATUS	MCMSTATUS (page 157)	Status information in each read block
CONTROL	MCMCONTROL (page 158)	Optional requests from the processor to the module
UTIL	Util (page 158)	Variables for internal ladder usage - should not be accessed by user application

6.7.2 MCMCONFIG

This object contains the data types that apply to the configuration of the module. Refer to MVI56E-MCM Configuration Data (page 144) for a complete description of each element in this object.

Name	Data Type	Description
ModDef	MCMModule (page 155)	Module Definition
Port1	MCMPort (page 155)	Port 1 configuration settings
Port2	MCMPort	Port 2 configuration settings
Port1MasterCmd	MCMCmd (page 156)	Master commands for Port 1 (ignore if port is configured for slave mode)

Name	Data Type	Description
Port2MasterCmd	MCMCmd[325]	Master commands for Port 2 (ignore if port is configured for slave mode)

Note: To use up to 325 commands, your MVI56E-MCM module needs to have firmware version 3.01 or higher, and your MVI56E-MCM Add-On Instruction needs to be version 2.8 or higher. Earlier versions support up to 100 commands.

MCMModule

This object contains the information used to define the data movement between the module and the processor.

Name	Data Type	Description
WriteStartReg	INT	Start reg to transfer from PLC to module
WriteRegCnt	INT	Number of registers to write from PLC
ReadStartReg	INT	Start reg to transfer from module to PLC
ReadRegCnt	INT	Number of registers to transfer from module
BPFail	INT	Determines module operation if BP fails 0=continue,>0=number of retries before comm shutdown
ErrStatPtr	INT	Internal DB start register for status data (-1=Ignore)

MCMPort

This object contains the serial port configuration for the MVI56E-MCM module.

Name	Data Type	Description
Enabled	INT	0=Port Disabled,1=Port Enabled
Type	INT	0=Master, 1=Slave, 2=Slave: pass-through, 3=Slave: formatted pass-through/data swapped, 4=Slave: form. pass-through
FloatFlag	INT	0=No floating-point data, 1=Use floating-point data
FloatStart	INT	Register offset in message for floating-point data
FloatOffset	INT	Internal DB offset to start of floating-point data
Protocol	INT	0=Modbus RTU, 1=Modbus ASCII
Baudrate	INT	Baudrate for port (110 to 115.2K)
Parity	INT	0=None, 1=Odd, 2=Even, 3=Mark, 4=Space
DataBits	INT	5 to 8 data bits
StopBits	INT	1 or 2 stop bits
RTSOn	INT	0-65535 mSec delay before data
RTSOff	INT	0-65535 mSec delay after data
MinResp	INT	0-65535 mSec minimum time before response to request

Name	Data Type	Description
UseCTS	INT	0=No, 1=Yes to use CTS modem line
SlaveID	INT	1-255 Modbus Node Address (Slave)
BitInOffset	INT	Internal DB offset to bit input data (Slave)
WordInOffset	INT	Internal DB offset to word input data (Slave)
OutOffset	INT	Internal DB offset to bit output data (Slave)
HoldOffset	INT	Internal DB offset to holding register data (Slave)
CmdCount	INT	Command list count (Master)
MinCmdDelay	INT	0-65535 mSec minimum time between each command (Master)
CmdErrPtr	INT	Internal DB location to place command error list (Master)
RespTO	INT	0-65535 mSec response timeout for command (Master)
RetryCount	INT	Retry count for failed request (Master)
ErrorDelayCntr	INT	0-65535 Command cycle count if error (Master)
Reserved	INT	Reserved (Previously was UseGuardBand parameter)
InterCharacterDelay	INT	0-65535 mSec time between characters to signal end of message
Fcn99Offset	INT	Internal DB offset to function 99 counter

MCMCmd

This object contains the attributes to define a Master command. An array of these objects is used for each port.

Name	Data Type	Description
Enable	INT	0=Disable,1=Continuous,2=Event Command
IntAddress	INT	Module's internal address associated with the command
PollInt	INT	Minimum number of seconds between issuance of command (0-65535 Sec)
Count	INT	Number of registers associated with the command
Swap	INT	Swap code used with command
Node	INT	Node address of the target device on the network
Func	INT	Function code for the command
DevAddress	INT	Address in device associated with the command. Hexadecimal format can be used to enter values above 32767.

6.7.3 MCMDATA

Contains Read Data (data read from the module to the processor) and Write Data (data written from the processor to the module).

Name	Data Type	Description
ReadData	INT[600]	Data read from the module to the processor
WriteData	INT[600]	Data written from the processor to the module

6.7.4 MCMSTATUS

This status data is returned on each read block and can be used to detect proper module operation.

Name	Data Type	Description
PassCnt	INT	Program cycle counter
Product	INT[2]	Product Name
Rev	INT[2]	Revision Level Number
OP	INT[2]	Operating Level Number
Run	INT[2]	Run Number
Prt1Errs	MCMPortErrors (page 157)	Port 1 error statistics
Prt2Errs	MCMPortErrors	Port 2 error statistics
Blk	MCMBlkStat (page 158)	Block transfer statistics
Port1LastErr	INT	Last command index that received an error on Port 1
Port1PreviousErr	INT	Previous Command index that received an error on Port 1
Port2LastErr	INT	Last command index that received an error on Port 2
Port2PreviousErr	INT	Previous Command index that received an error on Port 2

MCMPortErrors

This object stores the port statistics for an MVI56E-MCM port.

Name	Data Type	Description
CmdReq	INT	Total number of command list requests sent
CmdResp	INT	Total number of command list responses received
CmdErr	INT	Total number of command list errors
Requests	INT	Total number of requests for port
Responses	INT	Total number of responses for port
ErrSent	INT	Total number of errors sent
ErrRec	INT	Total number of errors received

MCMBlkStat

This object stores the block transfer statistics for the module.

Name	Data Type	Description
Read	INT	Total number of read block transfers
Write	INT	Total number of write block transfers
Parse	INT	Total number of blocks parsed
Event	INT	Total number of event blocks received
Cmd	INT	Total number of command blocks received
Err	INT	Total number of block transfer errors

6.7.5 MCMCONTROL

This object contains the attributes to define a Master command. An array of these objects is used for each port.

Name	Data Type	Description
WarmBoot	BOOL	Warm Boot
ColdBoot	BOOL	Cold Boot
BPLastRead	INT	Index of last read block
BPLastWrite	INT	Index of last write block
BlockIndex	INT	Computed block offset for data table
ReadDataSize	DINT	Size of Read Data Array
MaxReadBlock	DINT	Maximum read block
WriteDataSize	DINT	Size of Write Data Array
MaxWriteBlock	DINT	Maximum write block
RBTSremainder	INT	Contains remainder from Read Data array size divided by the block size
WBTSremainder	INT	Contains remainder from Write Data array size divided by the block size

6.7.6 Util

This object contains optional elements for the module.

Name	Data Type	Description
CmdControl	CmdControl (page 159)	Allows for a disabled command to be sent to a device (MASTER)
EventCmd	EventCmd (page 159)	Allows a command defined in ladder to be sent to a device (MASTER)
SlavePollStat	SlavePollStatus (page 159)	Request slave poll status for the port (MASTER).
Passthru	Passthru (page 160)	Contains PassThru objects required when PortX.Type is set to a value between 2 to 4.

CmdControl

Name	Data Type	Description
TriggerCmdCntrl	BOOL	Trigger command control. User application will activate this trigger
NumberOfCommands	INT	Number of commands per block (1 to 6)
PortNumber	INT	MVI56-MCM Port Number of master port (1 or 2)
CommandIndex	INT[6]	Stores the command indexes for command control
CmdsAddedToQueue	INT	Number of commands added to queue
CmdControlBlockID	INT	Temporary variable to calculate control block ID
CmdCntrlPending	BOOL	Auxiliary control command - prevents a second request before acknowledgment is received

EventCmd

Name	Data Type	Description
EventCmdTrigger	BOOL	Trigger for event command. User ladder must set this bit to initiate event command
EventCmdPending	BOOL	Set after the ladder has sent an event cmd to the module and is waiting for the status to be returned
PortNumber	INT	Module master port number associated to this request (1 or 2)
SlaveAddress	INT	Modbus slave node address
InternalDBAddress	INT	Internal database address
PointCount	INT	Number of points for this command
SwapCode	INT	Swap code (0= no swap, 1=swap words, 2=swap words and bytes, 3=swap bytes)
ModbusFunctionCode	INT	Modbus function code
DeviceDBAddress	INT	Modbus register address within slave
EventCmdStatusReturned	INT	(0=Fail, 1=Success)
EventBlockID	INT	Temporary variable to calculate event block ID

SlavePollStatus

This object contains all of the Slave Polling status (when the port is used as a Master).

Tag Name	Data Type	Description
Port1Slave0Read	BOOL	
Port1Slave128Read	BOOL	
Port2Slave0Read	BOOL	
Port2Slave128Read	BOOL	
P1Slaves	INT[256]	P1 Slave Status

Tag Name	Data Type	Description
P2Slaves	INT[256]	P2 Slave Status

Passthru

Tag Name	Data Type	Description
MBOffset	INT	
MBOffsetBit	INT	
MBMsgLen	INT	
MBMsg	SINT[500]	
MBControl1	MCMCONTROL (page 158)	
MBControl2	MCMCONTROL (page 158)	
MBScratch	INT[3]	
MBCoil	CoilArray (page 160)	Conversion from Bool to INT data types

CoilArray

Conversion to INT values

Name	Data Type	Description
Boolean	BOOL[416]	Conversion from Bool to INT data types

6.8 Modbus Protocol Specification

The following pages give additional reference information regarding the Modbus protocol commands supported by the MVI56E-MCM.

6.8.1 Commands Supported by the Module

The format of each command in the list depends on the Modbus Function Code being executed.

The following table lists the functions supported by the module.

Function Code	Definition	Supported in Master	Supported in Slave
1	Read Coil Status	X	X
2	Read Input Status	X	X
3	Read Holding Registers	X	X
4	Read Input Registers	X	X
5	Set Single Coil	X	X
6	Single Register Write	X	X
8	Diagnostics		X
15	Multiple Coil Write	X	X
16	Multiple Register Write	X	X
17	Report Slave ID		X

Function Code	Definition	Supported in Master	Supported in Slave
22	Mask Write 4X		X
23	Read/Write		X

Each command list record has the same general format. The first part of the record contains the information relating to the communication module and the second part contains information required to interface to the Modbus slave device.

6.8.2 Read Coil Status (Function Code 01)

Query

This function allows the user to obtain the ON/OFF status of logic coils used to control discrete outputs from the addressed Slave only. Broadcast mode is not supported with this function code. In addition to the Slave address and function fields, the message requires that the information field contain the initial coil address to be read (Starting Address) and the number of locations that is interrogated to obtain status data.

The addressing allows up to 2000 coils to be obtained at each request; however, the specific Slave device may have restrictions that lower the maximum quantity. The coils are numbered from zero; (coil number 1 = zero, coil number 2 = one, coil number 3 = two, and so on).

The following table is a sample read output status request to read coils 0020 to 0056 from Slave device number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display

Adr	Func	Data Start Pt Hi	Data Start Pt Lo	Data # Of Pts Ho	Data # Of Pts Lo	Error Check Field
0B	01	00	13	00	25	CRC

Response

An example response to Read Coil Status is as shown in the table below. The data is packed one bit for each coil. The response includes the Slave address, function code, quantity of data characters, the data characters, and error checking. Data is packed with one bit for each coil (1 = ON, 0 = OFF). The low order bit of the first character contains the addressed coil, and the remainder follow. For coil quantities that are not even multiples of eight, the last characters is filled in with zeros at high order end. The quantity of data characters is always specified as quantity of RTU characters, that is, the number is the same whether RTU or ASCII is used.

Because the Slave interface device is serviced at the end of a controller's scan, data reflects coil status at the end of the scan. Some Slaves limit the quantity of coils provided each scan; thus, for large coil quantities, multiple PC transactions must be made using coil status from sequential scans.

Adr	Func	Byte Count	Data Coil Status 20 to 27	Data Coil Status 28 to 35	Data Coil Status 36 to 43	Data Coil Status 44 to 51	Data Coil Status 52 to 56	Error Check Field
0B	01	05	CD	6B	B2	OE	1B	CRC

The status of coils 20 to 27 is shown as CD(HEX) = 1100 1101 (Binary). Reading left to right, this shows that coils 27, 26, 23, 22, and 20 are all on. The other coil data bytes are decoded similarly. Due to the quantity of coil statuses requested, the last data field, which is shown 1B (HEX) = 0001 1011 (Binary), contains the status of only 5 coils (52 to 56) instead of 8 coils. The 3 left most bits are provided as zeros to fill the 8-bit format.

6.8.3 Read Input Status (Function Code 02)

Query

This function allows the user to obtain the ON/OFF status of discrete inputs in the addressed Slave PC Broadcast mode is not supported with this function code. In addition to the Slave address and function fields, the message requires that the information field contain the initial input address to be read (Starting Address) and the number of locations that are interrogated to obtain status data.

The addressing allows up to 2000 inputs to be obtained at each request; however, the specific Slave device may have restrictions that lower the maximum quantity. The inputs are numbered form zero; (input 10001 = zero, input 10002 = one, input 10003 = two, and so on, for a 584).

The following table is a sample read input status request to read inputs 10197 to 10218 from Slave number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Adr	Func	Data Start Pt Hi	Data Start Pt Lo	Data #of Pts Hi	Data #of Pts Lo	Error Check Field
0B	02	00	C4	00	16	CRC

Response

An example response to Read Input Status is as shown in the table below. The data is packed one bit for each input. The response includes the Slave address, function code, quantity of data characters, the data characters, and error checking. Data is packed with one bit for each input (1=ON, 0=OFF). The lower order bit of the first character contains the addressed input, and the remainder follow. For input quantities that are not even multiples of eight, the last characters is filled in with zeros at high order end. The quantity of data characters is always specified as a quantity of RTU characters, that is, the number is the same whether RTU or ASCII is used.

Because the Slave interface device is serviced at the end of a controller's scan, data reflects input status at the end of the scan. Some Slaves limit the quantity of inputs provided each scan; thus, for large coil quantities, multiple PC transactions must be made using coil status for sequential scans.

Adr	Func	Byte Count	Data Discrete Input 10197 to 10204	Data Discrete Input 10205 to 10212	Data Discrete Input 10213 to 10218	Error Check Field
0B	02	03	AC	DB	35	CRC

The status of inputs 10197 to 10204 is shown as AC (HEX) = 10101 1100 (binary). Reading left to right, this show that inputs 10204, 10202, and 10199 are all on. The other input data bytes are decoded similar.

Due to the quantity of input statuses requested, the last data field which is shown as 35 HEX = 0011 0101 (binary) contains the status of only 6 inputs (10213 to 102180) instead of 8 inputs. The two left-most bits are provided as zeros to fill the 8-bit format.

6.8.4 Read Holding Registers (Function Code 03)

Query

Read Holding Registers (03) allows the user to obtain the binary contents of holding registers 4xxxx in the addressed Slave. The registers can store the numerical values of associated timers and counters which can be driven to external devices. The addressing allows up to 125 registers to obtained at each request; however, the specific Slave device may have restriction that lower this maximum quantity. The registers are numbered form zero (40001 = zero, 40002 = one, and so on). The broadcast mode is not allowed.

The example below reads registers 40108 through 40110 from Slave number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Adr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
0B	03	00	6B	00	03	CRC

Response

The addressed Slave responds with its address and the function code, followed by the information field. The information field contains 1 byte describing the quantity of data bytes to be returned. The contents of the registers requested (DATA) are two bytes each, with the binary content right justified within each pair of characters. The first byte includes the high order bits and the second, the low order bits.

Because the Slave interface device is normally serviced at the end of the controller's scan, the data reflects the register content at the end of the scan. Some Slaves limit the quantity of register content provided each scan; thus for large register quantities, multiple transmissions are made using register content from sequential scans.

In the example below, the registers 40108 to 40110 have the decimal contents 555, 0, and 100 respectively.

Adr	Func	ByteCnt	Hi Data	Lo Data	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
0B	03	06	02	2B	00	00	00	64	CRC

6.8.5 Read Input Registers (Function Code 04)

Query

Function code 04 obtains the contents of the controller's input registers from the Modbus 3x range. These locations receive their values from devices connected to the I/O structure and can only be referenced, not altered from within the controller, The addressing allows up to 125 registers to be obtained at each request; however, the specific Slave device may have restrictions that lower this maximum quantity. The registers are numbered for zero (30001 = zero, 30002 = one, and so on). Broadcast mode is not allowed.

The example below requests the contents of register 3009 in Slave number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Adr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
0B	04	00	08	00	01	CRC

Response

The addressed Slave responds with its address and the function code followed by the information field. The information field contains 1 byte describing the quantity of data bytes to be returned. The contents of the registers requested (DATA) are 2 bytes each, with the binary content right justified within each pair of characters. The first byte includes the high order bits and the second, the low order bits.

Because the Slave interface is normally serviced at the end of the controller's scan, the data reflects the register content at the end of the scan. Each PC limits the quantity of register contents provided each scan; thus for large register quantities, multiple PC scans are required, and the data provided is from sequential scans.

In the example below the register 3009 contains the decimal value 0.

Adr	Func	Byte Count	Data Input Reg Hi	Data Input Reg Lo	Error Check Field
0B	04	02	00	00	E9

6.8.6 Force Single Coil (Function Code 05)

Query

This Function Code forces a single coil (Modbus 0x range) either ON or OFF. Any coil that exists within the controller can be forced to either state (ON or OFF). However, because the controller is actively scanning, unless the coil is disabled, the controller can also alter the state of the coil. Coils are numbered from zero (coil 0001 = zero, coil 0002 = one, and so on). The data value 65,280 (FF00 HEX) sets the coil ON and the value zero turns it OFF; all other values are illegal and does not affect that coil.

The use of Slave address 00 (Broadcast Mode) forces all attached Slaves to modify the desired coil.

Note: Functions 5, 6, 15, and 16 are the only messages that are recognized as valid for broadcast.

The example below is a request to Slave number 11 to turn ON coil 0173.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Adr	Func	Data Coil # Hi	Data Coil # Lo	Data On/off Ind	Data	Error Check Field
0B	05	00	AC	FF	00	CRC

Response

The normal response to the Command Request is to re-transmit the message as received after the coil state has been altered.

Adr	Func	Data Coil # Hi	Data Coil # Lo	Data On/ Off	Data	Error Check Field
0B	05	00	AC	FF	00	CRC

The forcing of a coil via Modbus function 5 is accomplished regardless of whether the addressed coil is disabled or not (*In ProSoft products,* the coil *is only affected if the necessary ladder logic is implemented).*

Note: The Modbus protocol does not include standard functions for testing or changing the DISABLE state of discrete inputs or outputs. Where applicable, this may be accomplished via device specific Program commands (*In ProSoft products, this is only accomplished through ladder logic programming).*

Coils that are reprogrammed in the controller logic program are not automatically cleared upon power up. Thus, if such a coil is set ON by function Code 5 and (even months later), an output is connected to that coil, the output is "hot".

6.8.7 Preset Single Register (Function Code 06)

Query

This Function Code allows you to modify the contents of a Modbus 4x range in the Slave. This writes to a single register only. Any holding register that exists within the controller can have its contents changed by this message. However, because the controller is actively scanning, it also can alter the content of any holding register at any time. The values are provided in binary up to the maximum capacity of the controller. Unused high order bits must be set to zero. When used with Slave address zero (Broadcast mode) all Slave controllers will load the specified register with the contents specified.

Note Functions 5, 6, 15, and 16 are the only messages that will be recognized as valid for broadcast.
Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

The example below is a request to write the value ‘3’ to register 40002 in slave 11.

Adr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
0B	06	00	01	00	03	CRC

Response

The response to a preset single register request is to re-transmit the query message after the register has been altered.

Adr	Func	Data Reg Hi	Data Reg Lo	Data Input Reg Hi	Data Input Reg Lo	Error Check Field
0B	06	00	01	00	03	CRC

6.8.8 Diagnostics (Function Code 08)

Modbus function code 08 provides a series of tests for checking the communication system between a Master device and a slave, or for checking various internal error conditions within a slave.

The function uses a two-byte sub-function code field in the query to define the type of test to be performed. The slave echoes both the function code and sub-function code in a normal response. Some of the diagnostics commands cause data to be returned from the remote device in the data field of a normal response.

In general, issuing a diagnostic function to a remote device does not affect the running of the user program in the remote device. Device memory bit and register data addresses are not accessed by the diagnostics. However, certain functions can optionally reset error counters in some remote devices.

A server device can, however, be forced into 'Listen Only Mode' in which it will monitor the messages on the communications system but not respond to them. This can affect the outcome of your application program if it depends upon any further exchange of data with the remote device. Generally, the mode is forced to remove a malfunctioning remote device from the communications system.

Sub-function Codes Supported

Only Sub-function 00 is supported by the MVI56E-MCM module.

00 Return Query Data

The data passed in the request data field is to be returned (looped back) in the response. The entire response message should be identical to the request.

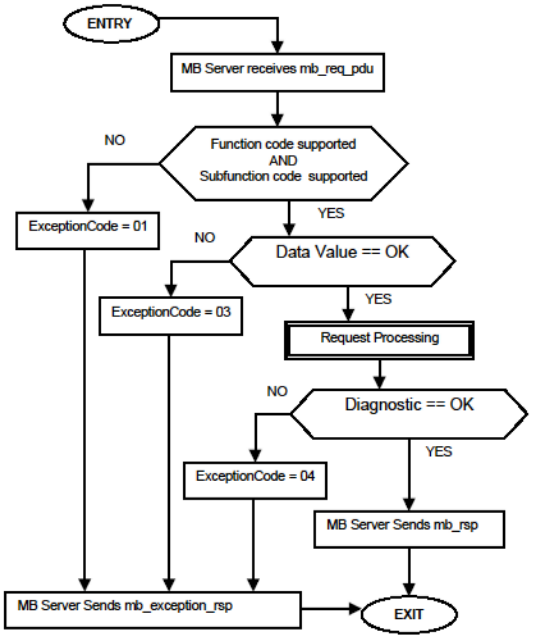
Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

Example and State Diagram

Here is an example of a request to remote device to Return Query Data. This uses a sub-function code of zero (00 00 hex in the two-byte field). The data to be returned is sent in the two-byte data field (A5 37 hex).

Request		Response	
Field Name	**(Hex)**	**Field Name**	**(Hex)**
Function	08	Function	08
Sub-function Hi	00	Sub-function Hi	00
Sub-function Lo	00	Sub-function Lo	00
Data Hi	A5	Data Hi	A5
Data Lo	37	Data Lo	27

The data fields in responses to other kinds of queries could contain error counts or other data requested by the sub-function code.



6.8.9 Force Multiple Coils (Function Code 15)

Query

This Function Code forces each coil (Modbus 0x range) in a consecutive block of coils to a desired ON or OFF state. Any coil that exists within the controller can be forced to either state (ON or OFF). However, because the controller is actively scanning, unless the coils are disabled, the controller can also alter the state of the coil. Coils are numbered from zero (coil 00001 = zero, coil 00002 = one, and so on). The desired status of each coil is packed in the data field, one bit for each coil (1= ON, 0= OFF). The use of Slave address 0 (Broadcast Mode) will force all attached Slaves to modify the desired coils.

Note: Functions 5, 6, 15, and 16 are the only messages (other than Loopback Diagnostic Test) that will be recognized as valid for broadcast.

The following example forces 10 coils starting at address 20 (13 HEX). The two data fields, CD =1100 and 00 = 0000 000, indicate that coils 27, 26, 23, 22, and 20 are to be forced on.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Adr	Func	Hi Add	Lo Add	Quantity	Byte Cnt	Data Coil Status 20 to 27	Data Coil Status 28 to 29	Error Check Field	
0B	0F	00	13	00	0A	02	CD	00	CRC

Response

The normal response will be an echo of the Slave address, function code, starting address, and quantity of coils forced.

Adr	Func	Hi Addr	Lo Addr	Quantity	Error Check Field	
0B	0F	00	13	00	0A	CRC

The writing of coils via Modbus function 15 will be accomplished regardless of whether the addressed coils are disabled or not.

Coils that are not programmed in the controller logic program are not automatically cleared upon power up. Thus, if such a coil is set ON by function code 15 and (even months later) an output is connected to that coil, the output is hot.

6.8.10 Preset Multiple Registers (Function Code 16)

Query

Holding registers existing within the controller can have their contents changed by this message (a maximum of 60 registers). However, because the controller is actively scanning, it also can alter the content of any holding register at any time. The values are provided in binary up to the maximum capacity of the controller (16-bit for the 184/384 and 584); unused high order bits must be set to zero.

Note: Function codes 5, 6, 15, and 16 are the only messages that will be recognized as valid for broadcast.

Adr	Func	Hi Add	Lo Add	Quantity		Byte Cnt	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
11	10	00	87	00	02	04	00	0A	01	02	CRC

Response

The normal response to a function 16 query is to echo the address, function code, starting address and number of registers to be loaded.

Adr	Func	Hi Addr	Lo Addr	Quantity		Error Check Field
11	10	00	87	00	02	56

6.8.11 Modbus Exception Responses

When a Modbus Master sends a request to a Slave device, it expects a normal response. One of four possible events can occur from the Master's query:

- If the server device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the server does not receive the request due to a communication error, no response is returned. The Master program will eventually process a timeout condition for the request.
- If the server receives the request, but detects a communication error (parity, LRC, CRC, ...), no response is returned. The Master program will eventually process a timeout condition for the request.
- If the server receives the request without a communication error, but cannot handle it (for example, if the request is to read a non-existent output or register), the server will return an exception response informing the Master of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function Code Field: In a normal response, the server echoes the function code of the original request in the function code field of the response. All function codes have a most-significant bit (MSB) of 0 (their values are all below 80 hexadecimal). In an exception response, the server sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response.

With the function code's MSB set, the Master's application program can recognize the exception response and can examine the data field for the exception code.

Data Field: In a normal response, the server may return data or statistics in the data field (any information that was requested in the request). In an exception response, the server returns an exception code in the data field. This defines the server condition that caused the exception.

The following table shows an example of a Master request and server exception response.

Request		Response	
Field Name	**(Hex)**	**Field Name**	**(Hex)**
Function	01	Function	81
Starting Address Hi	04	Exception Code	02
Starting Address Lo	A1		
Quantity of Outputs Hi	00		
Quantity of Outputs Lo	01		

In this example, the Master addresses a request to server device. The function code (01) is for a Read Output Status operation. It requests the status of the output at address 1245 (04A1 hex). Note that only that one output is to be read, as specified by the number of outputs field (0001).

If the output address is non-existent in the server device, the server will return the exception response with the exception code shown (02). This specifies an illegal data address for the Slave.

Modbus Exception Codes

Code	Name	Meaning
01	Illegal Function	The function code received in the query is not an allowable action for the Slave. This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the Slave is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.

Code	Name	Meaning
02	Illegal Data Address	The data address received in the query is not an allowable address for the Slave. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed; a request with offset 96 and length 5 will generate exception 02.
03	Illegal Data Value	A value contained in the query data field is not an allowable value for Slave. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expectation of the application program, because the Modbus protocol is unaware of the significance of any particular value of any particular register.
04	Slave Device Failure	An unrecoverable error occurred while the Slave was attempting to perform the requested action.
05	Acknowledge	Specialized use in conjunction with programming commands. The Slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the Master. The Master can next issue a poll program complete message to determine if processing is completed.
06	Slave Device Busy	Specialized use in conjunction with programming commands. The Slave is engaged in processing a long-duration program command. The Master should retransmit the message later when the Slave is free.

6.9 Using the Optional Add-On Instruction

6.9.1 Before You Begin

- Make sure that you have installed RSLogix 5000 version 16 (or later).
- Download the Optional Add-On file *MVI56(E)MCM_Optional_AddOn_Rung_v1_2.L5X* from www.prosoft-technology.com or copy it from the *ProSoft Solutions DVD*.
- Save a copy in a folder in your PC.

6.9.2 Overview

The Optional Add-On Instruction Rung Import contains optional logic for MVI56E-MCM applications to perform the following tasks.

- Read/Write Ethernet Configuration

  Allows the processor to read or write the module IP address, netmask and gateway values.

Note: This is an optional feature. You can perform the same task through PCB (ProSoft Configuration Builder). Even if your PC is in a different network group you can still access the module through PCB by setting a temporary IP address.

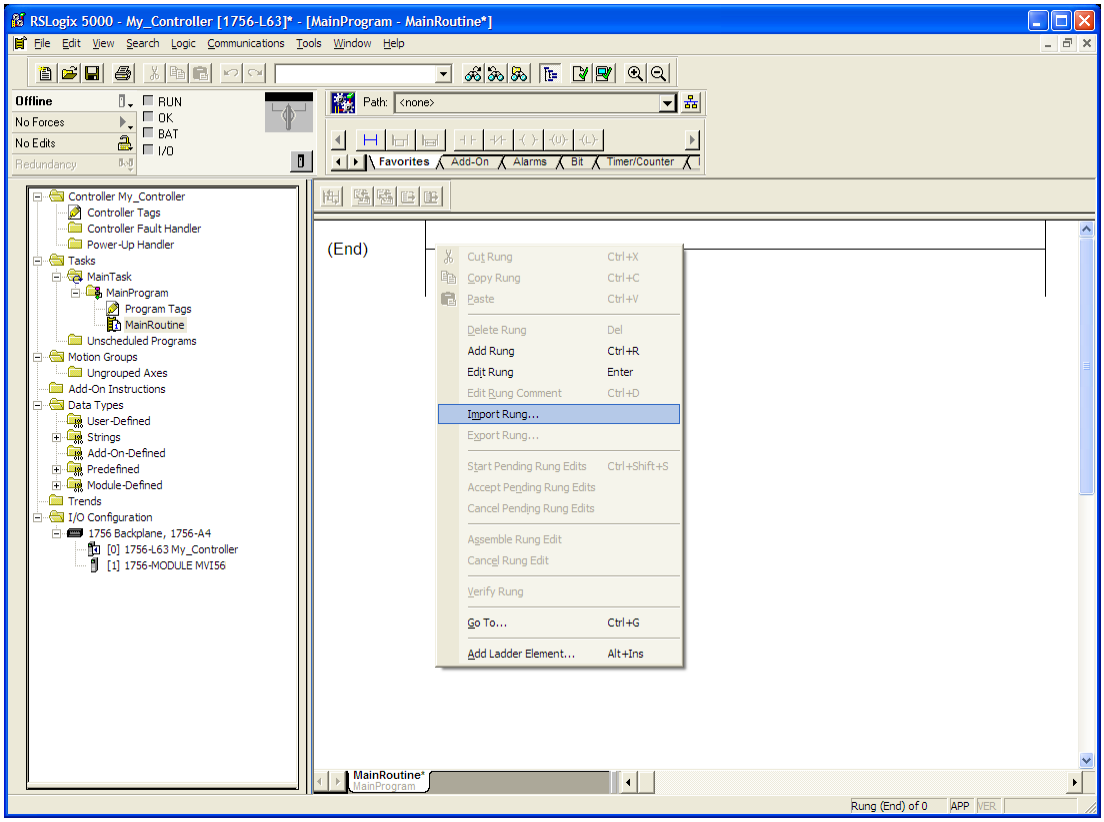
- Read/Write Module Clock Value

  Allows the processor to read and write the module clock settings. The module clock stores the last time that the Ethernet configuration was changed. The date and time of the last Ethernet configuration change is displayed in the scrolling LED during module power up.

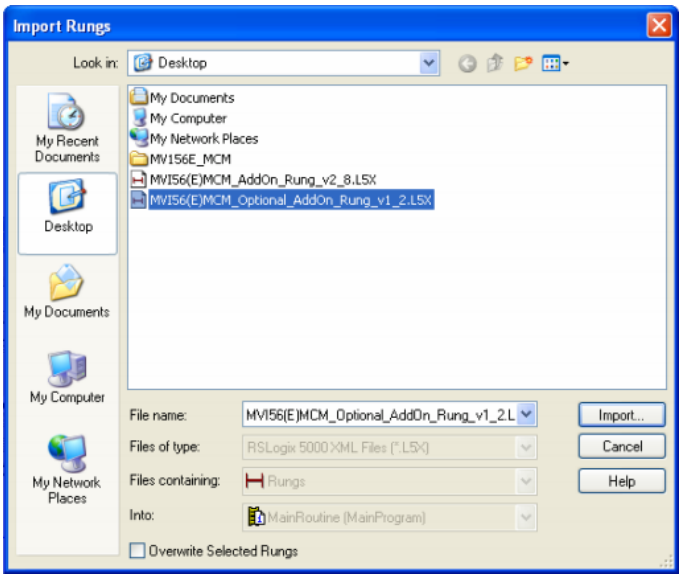
Important: The Optional Add-On Instruction only supports the two features listed above. You must use the sample ladder logic for all other features including backplane transfer of Modbus data.

6.9.3 Importing the Utility Add-On Instruction

1 Right-click on an empty rung in the main routine of your existing ladder logic and choose **IMPORT RUNGS…**

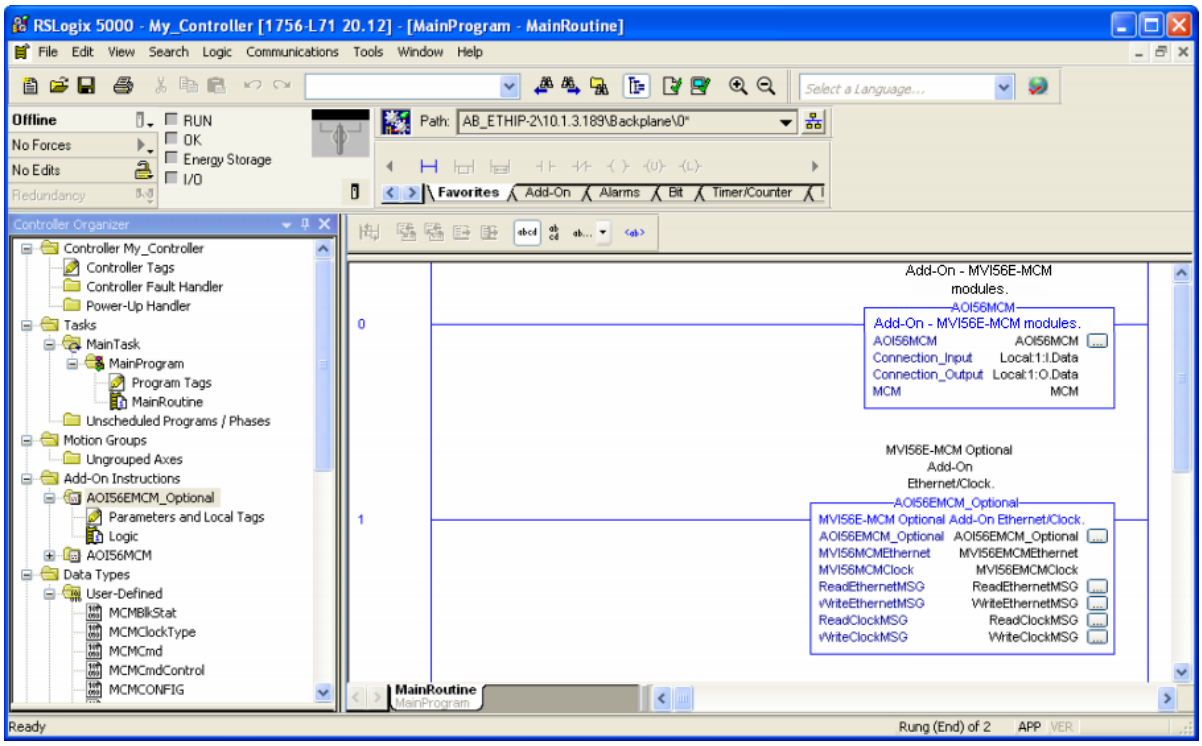


2 Navigate to the folder where you saved MVI56(E)MCM_Optional_AddOn_Rung_v1_2.L5X and select the file.

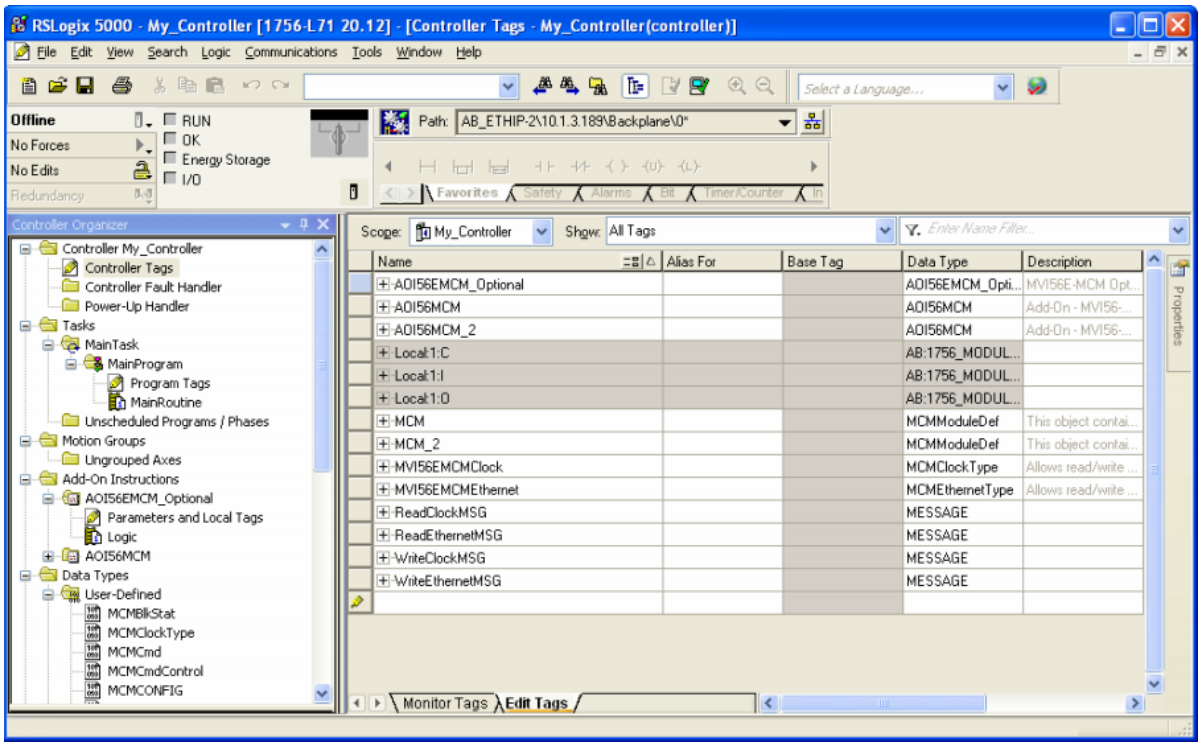


3 In the **IMPORT CONFIGURATION** window, click **OK.**

The Add-On Instruction is now visible in the ladder logic. Observe that the procedure has also imported data types and controller tags associated to the Add-On Instruction.

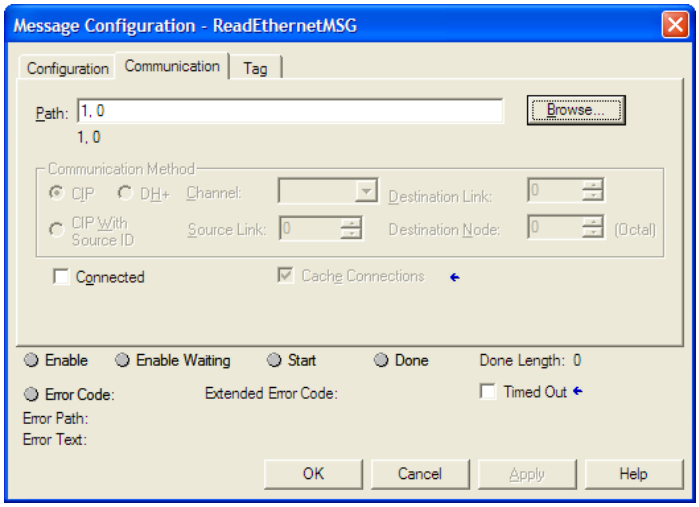


You will notice that new tags have been imported: four **MESSAGE** tags, **MVI56MCMCLOCK** and **MVI56MCMETHERNET** tags.

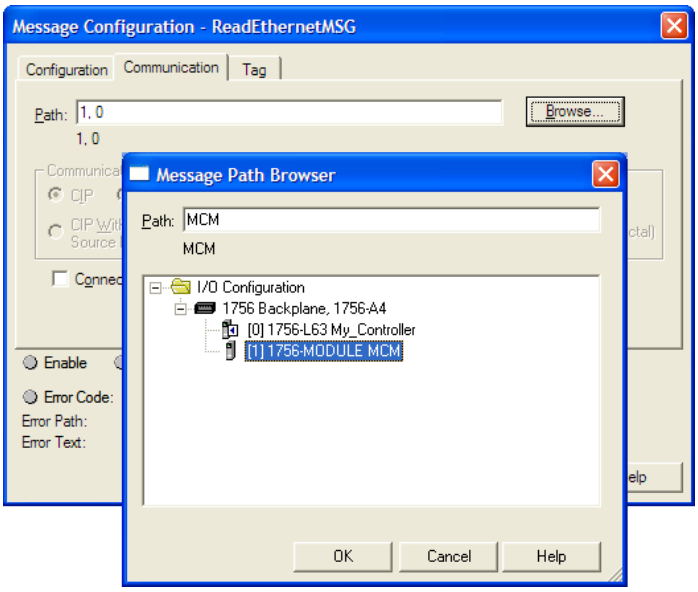


4 In the Add-On Instruction click the **[...]** button next to each **MSG** tag to open the **MESSAGE CONFIGURATION TAG**.

5 Click the **COMMUNICATION** tab and click the **BROWSE** button as follows.



6 Select the module to configure the message path.



6.9.4 Reading the Ethernet Settings from the Module

Expand the **MVI56MCMETHERNET** controller tag and move a value of 1 to **MVI56MCMETHERNET.READ.**

MVI56MCMEthernet	{...}
MVI56MCMEthernet.Read	1
MVI56MCMEthernet.Write	0
MVI56MCMEthernet.Config	{...}
MVI56MCMEthernet.Config.IP	{...}
MVI56MCMEthernet.Config.IP[0]	0
MVI56MCMEthernet.Config.IP[1]	0
MVI56MCMEthernet.Config.IP[2]	0
MVI56MCMEthernet.Config.IP[3]	0
MVI56MCMEthernet.Config.Netmask	{...}
MVI56MCMEthernet.Config.Netmask[0]	0
MVI56MCMEthernet.Config.Netmask[1]	0
MVI56MCMEthernet.Config.Netmask[2]	0
MVI56MCMEthernet.Config.Netmask[3]	0
MVI56MCMEthernet.Config.Gateway	{...}
MVI56MCMEthernet.Config.Gateway[0]	0
MVI56MCMEthernet.Config.Gateway[1]	0
MVI56MCMEthernet.Config.Gateway[2]	0
MVI56MCMEthernet.Config.Gateway[3]	0

The bit will be automatically reset and the current Ethernet settings will be copied to **MVI56MCMETHERNET** controller tag as follows.

MVI56MCMEthernet	{...}
MVI56MCMEthernet.Read	0
MVI56MCMEthernet.Write	0
MVI56MCMEthernet.Config	{...}
MVI56MCMEthernet.Config.IP	{...}
MVI56MCMEthernet.Config.IP[0]	105
MVI56MCMEthernet.Config.IP[1]	102
MVI56MCMEthernet.Config.IP[2]	0
MVI56MCMEthernet.Config.IP[3]	132
MVI56MCMEthernet.Config.Netmask	{...}
MVI56MCMEthernet.Config.Netmask[0]	255
MVI56MCMEthernet.Config.Netmask[1]	255
MVI56MCMEthernet.Config.Netmask[2]	255
MVI56MCMEthernet.Config.Netmask[3]	0
MVI56MCMEthernet.Config.Gateway	{...}
MVI56MCMEthernet.Config.Gateway[0]	192
MVI56MCMEthernet.Config.Gateway[1]	168
MVI56MCMEthernet.Config.Gateway[2]	0
MVI56MCMEthernet.Config.Gateway[3]	1

To check the status of the message, refer to the **READETHERNETMSG** tag.

ReadEthernetMSG	{...}
ReadEthernetMSG.Flags	16#0220
ReadEthernetMSG.EW	0
ReadEthernetMSG.ER	0
ReadEthernetMSG.DN	1
ReadEthernetMSG.ST	0
ReadEthernetMSG.EN	0
ReadEthernetMSG.TO	0
ReadEthernetMSG.EN_CC	1
ReadEthernetMSG.ERR	16#0000
ReadEthernetMSG.EXERR	16#0000_0000
ReadEthernetMSG.ERR_SRC	0
ReadEthernetMSG.DN_LEN	24

6.9.5 Writing the Ethernet Settings to the Module

Expand the **MVI56MCMETHERNET** controller tag.

Set the new Ethernet configuration in **MVI56MCMETHERNET.CONFIG**

Move a value of 1 to **MVI56MCMETHERNET.WRITE**

MVI56MCMEthernet	{...}
MVI56MCMEthernet.Read	0
MVI56MCMEthernet.Write	1
MVI56MCMEthernet.Config	{...}
MVI56MCMEthernet.Config.IP	{...}
MVI56MCMEthernet.Config.IP[0]	105
MVI56MCMEthernet.Config.IP[1]	102
MVI56MCMEthernet.Config.IP[2]	0
MVI56MCMEthernet.Config.IP[3]	132
MVI56MCMEthernet.Config.Netmask	{...}
MVI56MCMEthernet.Config.Netmask[0]	255
MVI56MCMEthernet.Config.Netmask[1]	255
MVI56MCMEthernet.Config.Netmask[2]	255
MVI56MCMEthernet.Config.Netmask[3]	0
MVI56MCMEthernet.Config.Gateway	{...}
MVI56MCMEthernet.Config.Gateway[0]	192
MVI56MCMEthernet.Config.Gateway[1]	168
MVI56MCMEthernet.Config.Gateway[2]	0
MVI56MCMEthernet.Config.Gateway[3]	1

After the message is executed, the **MVI56MCMETHERNET.WRITE** bit resets to 0.

MVI56MCMEthernet	{...}
MVI56MCMEthernet.Read	0
MVI56MCMEthernet.Write	0
MVI56MCMEthernet.Config	{...}
MVI56MCMEthernet.Config.IP	{...}
MVI56MCMEthernet.Config.IP[0]	105
MVI56MCMEthernet.Config.IP[1]	102
MVI56MCMEthernet.Config.IP[2]	0
MVI56MCMEthernet.Config.IP[3]	132
MVI56MCMEthernet.Config.Netmask	{...}
MVI56MCMEthernet.Config.Netmask[0]	255
MVI56MCMEthernet.Config.Netmask[1]	255
MVI56MCMEthernet.Config.Netmask[2]	255
MVI56MCMEthernet.Config.Netmask[3]	0
MVI56MCMEthernet.Config.Gateway	{...}
MVI56MCMEthernet.Config.Gateway[0]	192
MVI56MCMEthernet.Config.Gateway[1]	168
MVI56MCMEthernet.Config.Gateway[2]	0
MVI56MCMEthernet.Config.Gateway[3]	1

To check the status of the message, refer to the **WRITEETHERNETMSG** tag.

WriteEthernetMSG	{...}
WriteEthernetMSG.Flags	16#0220
WriteEthernetMSG.EW	0
WriteEthernetMSG.ER	0
WriteEthernetMSG.DN	1
WriteEthernetMSG.ST	0
WriteEthernetMSG.EN	0
WriteEthernetMSG.TO	0
WriteEthernetMSG.EN_CC	1
WriteEthernetMSG.ERR	16#0000
WriteEthernetMSG.EXERR	16#0000_0000
WriteEthernetMSG.ERR_SRC	0
WriteEthernetMSG.DN_LEN	0
WriteEthernetMSG.REQ_LEN	24

6.9.6 Reading the Clock Value from the Module

Expand the **MVI56MCMCLOCK** controller tag and move a value of 1 to **MVI56MCMCLOCK.READ**

MVI56MCMClock	{...}
MVI56MCMClock.Read	1
MVI56MCMClock.Write	0
MVI56MCMClock.Config	{...}
MVI56MCMClock.Config.Year	0
MVI56MCMClock.Config.Month	0
MVI56MCMClock.Config.Day	0
MVI56MCMClock.Config.Hour	0
MVI56MCMClock.Config.Minute	0
MVI56MCMClock.Config.Seconds	0

The bit will be automatically reset and the current clock value will be copied to **MVI56MCMCLOCK.CONFIG** controller tag as follows.

MVI56MCMClock	{...}
MVI56MCMClock.Read	0
MVI56MCMClock.Write	0
MVI56MCMClock.Config	{...}
MVI56MCMClock.Config.Year	2008
MVI56MCMClock.Config.Month	11
MVI56MCMClock.Config.Day	12
MVI56MCMClock.Config.Hour	15
MVI56MCMClock.Config.Minute	38
MVI56MCMClock.Config.Seconds	9

To check the status of the message, refer to the **READCLOCKMSG** tag.

ReadClockMSG	{...}
ReadClockMSG.Flags	16#0220
ReadClockMSG.EW	0
ReadClockMSG.ER	0
ReadClockMSG.DN	1
ReadClockMSG.ST	0
ReadClockMSG.EN	0
ReadClockMSG.TO	0
ReadClockMSG.EN_CC	1
ReadClockMSG.ERR	16#0000
ReadClockMSG.EXERR	16#0000_0000
ReadClockMSG.ERR_SRC	0
ReadClockMSG.DN_LEN	24

6.9.7 Writing the Clock Value to the Module

Expand the **MVI56MCMCLOCK** controller tag.

Set the new Clock value in **MVI56MCMCLOCK.CONFIG**

Move a value of 1 to **MVI56MCMCLOCK.WRITE**

MVI56MCMClock	{...}
MVI56MCMClock.Read	0
MVI56MCMClock.Write	1
MVI56MCMClock.Config	{...}
MVI56MCMClock.Config.Year	2008
MVI56MCMClock.Config.Month	11
MVI56MCMClock.Config.Day	12
MVI56MCMClock.Config.Hour	15
MVI56MCMClock.Config.Minute	38
MVI56MCMClock.Config.Seconds	9

The bit will be automatically reset to 0.

MVI56MCMClock	{...}
MVI56MCMClock.Read	0
MVI56MCMClock.Write	0
MVI56MCMClock.Config	{...}
MVI56MCMClock.Config.Year	2008
MVI56MCMClock.Config.Month	11
MVI56MCMClock.Config.Day	12
MVI56MCMClock.Config.Hour	15
MVI56MCMClock.Config.Minute	38
MVI56MCMClock.Config.Seconds	9

To check the status of the message, refer to the **WRITECLOCKMSG** tag.

WriteClockMSG	{...}
WriteClockMSG.Flags	16#0220
WriteClockMSG.EW	0
WriteClockMSG.ER	0
WriteClockMSG.DN	1
WriteClockMSG.ST	0
WriteClockMSG.EN	0
WriteClockMSG.TO	0
WriteClockMSG.EN_CC	1
WriteClockMSG.ERR	16#0000
WriteClockMSG.EXERR	16#0000_0000
WriteClockMSG.ERR_SRC	0
WriteClockMSG.DN_LEN	0
WriteClockMSG.REQ_LEN	24

6.10 Using the Sample Program - RSLogix 5000 Version 15 and earlier

The sample program included with your MVI56E-MCM module contains predefined controller tags, configuration information, data types, and ladder logic that allow the module to communicate between the ControlLogix processor and a network of Modbus devices. For most applications, the sample program will work without modification.

6.10.1 Using the Sample Program in a New Application

Opening the Sample Program in RSLogix

The sample program for your MVI56E-MCM module includes custom tags, data types and ladder logic for data I/O, status and command control. For most applications, you can run the sample program without modification, or, for advanced applications, you can incorporate the sample program into your existing application.

The ProSoft Solutions DVD provides one or more versions of the sample ladder logic. The version number appended to the file name corresponds with the firmware version number of your ControlLogix processor. The firmware version and sample program version must match.

The ProSoft Solutions DVD included in the package with the module contains ladder logic, product manuals and utility programs for all ProSoft Technology products.

Copy the manuals and sample program from the DVD

1. Insert the ProSoft Solutions DVD into the DVD drive of your PC. Wait for the startup screen to appear.
2. On the startup screen, click *Product Documentation*. This action opens an Explorer window. Files are arranged by type:
   - The Ladder Logic folder contains sample programs for each module, arranged by processor type, and then by product name. The sample programs for your module are in the ControlLogix/MVI56/MVI56E-MCM folder.
   - The Manuals folder contains product manuals and datasheets in Adobe Acrobat Reader format (PDF) for each module, arranged in the same way as the Ladder Logic folder.
   - The Utilities folder contains additional programs and tools required for some ProSoft modules. Refer to your user manual to determine if you need to use or install any of these additional tools.
3. In the Explorer window, navigate to the files you need, and then copy them to a location on your hard drive.

Download the manuals and sample program from the ProSoft Technology web site

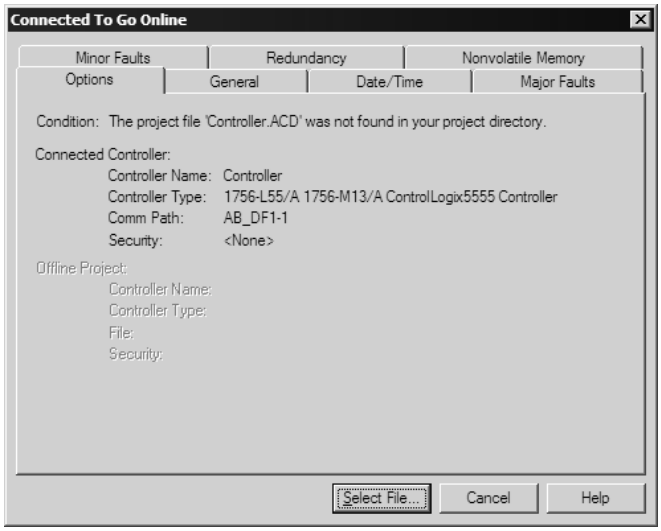
You can always download the latest version of the sample ladder logic and user manuals for the MVI56E-MCM module from the ProSoft Technology web site: http://www.prosoft-technology.com/prosoft/support/downloads

From that link, navigate to the download page for your module and choose the sample program to download for your version of RSLogix 5000 and your processor.

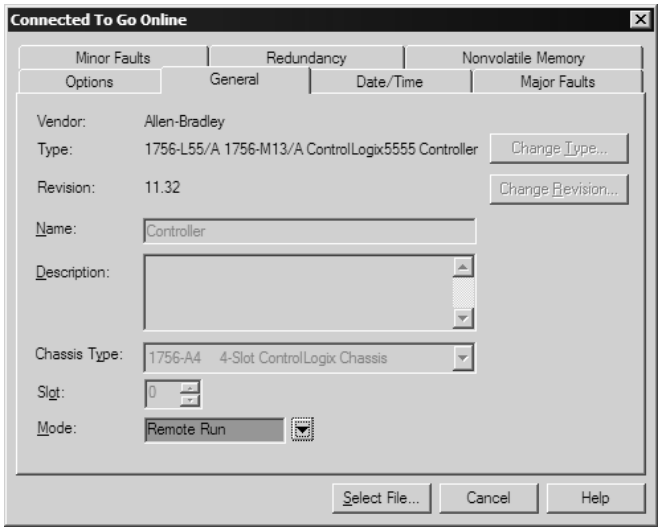
To determine the firmware version of your processor

Important: The RSLinx service must be installed and running on your computer in order for RSLogix to communicate with the processor. Refer to your RSLinx and RSLogix documentation for help configuring and troubleshooting these applications.

1. Connect an RS-232 serial cable from the COM (serial) port on your PC to the communication port on the front of the processor.
2. Start RSLogix 5000 and close any existing project that may be loaded.
3. Open the Communications menu and choose **Go Online**. RSLogix will establish communication with the processor. This may take a few moments.
4. When RSLogix has established communication with the processor, the Connected To Go Online dialog box will open.



5. On the Connected To Go Online dialog box, click the General tab. This tab shows information about the processor, including the Revision (firmware) version. In the following illustration, the firmware version is 11.32



6 Select the sample ladder logic file for your firmware version:

To open the sample program

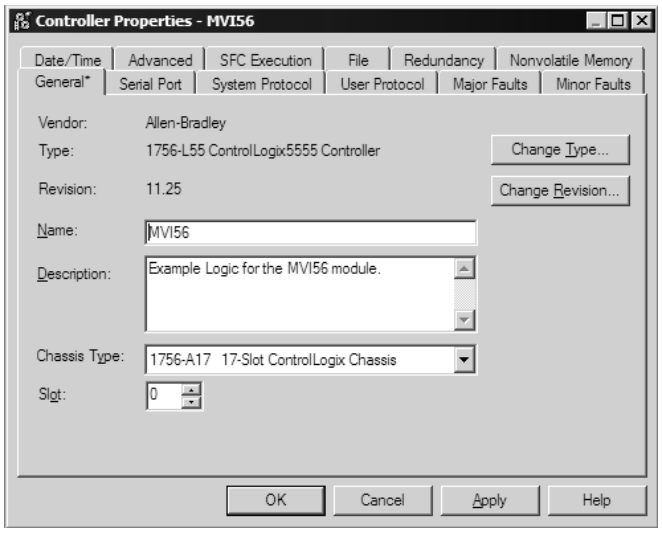
1 On the Connected to Go Online dialog box, click the Select File button.
2 Choose the sample program file that matches your firmware version, and then click the Select button.
3 RSLogix will load the sample program.

The next step is to configure the correct controller type and slot number for your application.

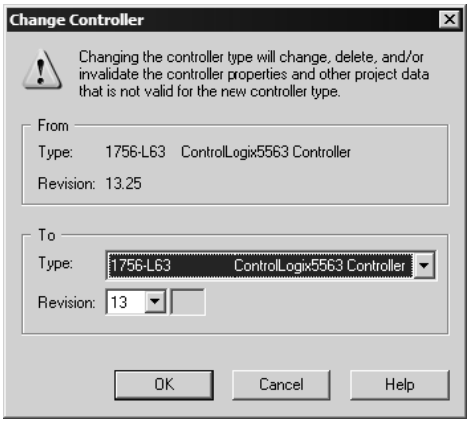
Choosing the Controller Type

The sample application is for a 1756-L63 ControlLogix 5563 Controller. If you are using a different model of the ControlLogix processor, you must configure the sample program to use the correct processor model.

1 In the *Controller Organization* list, select the folder for the controller and then click the right mouse button to open a shortcut menu.
2 On the shortcut menu, choose **PROPERTIES.** This action opens the *Controller Properties* dialog box.



3 Click the **CHANGE TYPE** or **CHANGE CONTROLLER** button. This action opens the *Change Controller* dialog box.


Change Controller
Changing the controller type will change, delete, and/or invalidate the controller properties and other project data that is not valid for the new controller type.
From
Type: 1756-L63 ControlLogix5563 Controller
Revision: 13.25
To
Type: 1756-L63 ControlLogix5563 Controller
Revision: 13
OK
Cancel
Help

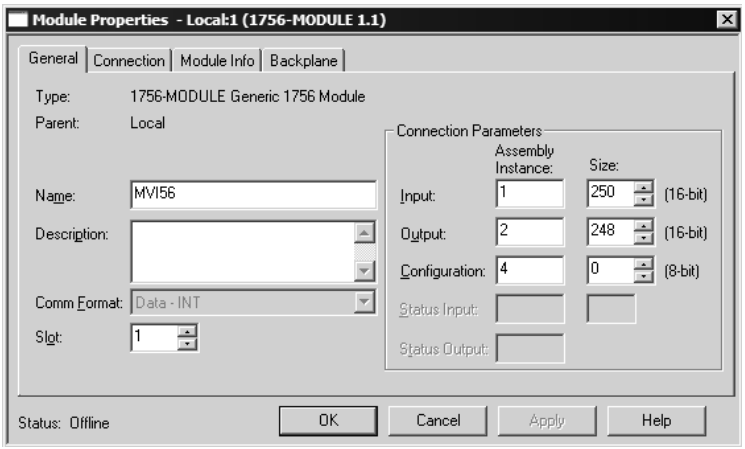
4 Open the **TYPE** dropdown list, and then select your ControlLogix controller.
5 Select the correct firmware revision for your controller, if necessary.
6 Click **OK** to save your changes and return to the previous window.

Selecting the Slot Number for the Module

The sample application is for a module installed in Slot 1 in a ControlLogix rack. The ladder logic uses the slot number to identify the module. If you are installing the module in a different slot, you must update the ladder logic so that program tags and variables are correct, and do not conflict with other modules in the rack.

To change the slot number

1 In the **CONTROLLER ORGANIZATION** list, select the module **[1] 1756-MODULE MVI56,** and then click the right mouse button to open a shortcut menu.
2 On the shortcut menu, choose **PROPERTIES.** This action opens the **MODULE PROPERTIES** dialog box.



3 In the **SLOT** field, use the up and down arrows on the right side of the field to select the slot number where the module will reside in the rack, and then click **OK.**

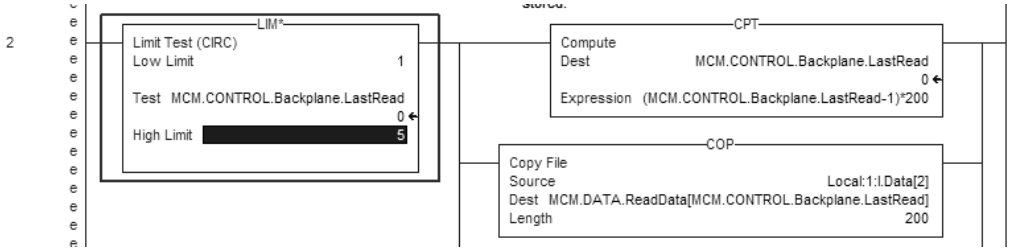
RSLogix will automatically apply the slot number change to all tags, variables and ladder logic rungs that use the MVI56E-MCM slot number for computation.

Adjust the Input and Output Array Sizes

Note: The following steps are only required if you are using the sample ladder logic (RSLogix version 15 or older) rather than the Add-On Instruction (RSLogix version 16 or newer).

1. Click **ReadData** to open ladder file and go to rung #2 of this file.
2. Change the High Limit on the **LIM** statement to allow for 5 blocks of data, as shown in the following illustration.

   (1000 registers / 200 registers per block = 5 blocks of data)

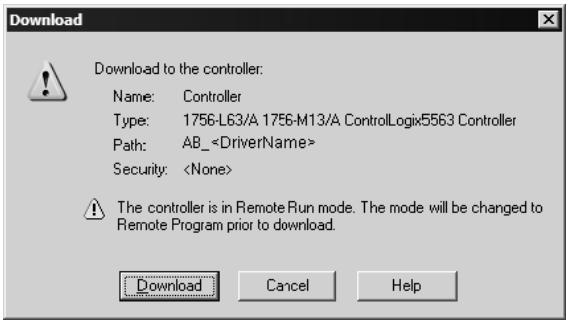


3. Verify the change to this rung. Toggle the object within RSLogix 5000.
4. Save and download ladder to the processor.
5. When Online with the ControlLogix processor, toggle the **MCM.CONTROL.WARMBOOT** bit to download the changes to the processor.

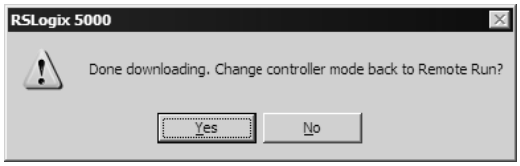
Downloading the Sample Program to the Processor

Note: The key switch on the front of the ControlLogix module must be in the REM position.

1. If you are not already online to the processor, open the **COMMUNICATIONS** menu, and then choose **DOWNLOAD**. RSLogix will establish communication with the processor.
2. When communication is established, RSLogix will open a confirmation dialog box. Click the **DOWNLOAD** button to transfer the sample program to the processor.



3 RSLogix will compile the program and transfer it to the processor. This process may take a few minutes.
4 When the download is complete, RSLogix will open another confirmation dialog box. Click **OK** to switch the processor from PROGRAM mode to RUN mode.



Note: If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

6.10.2 Using the Sample Program in an Existing Application

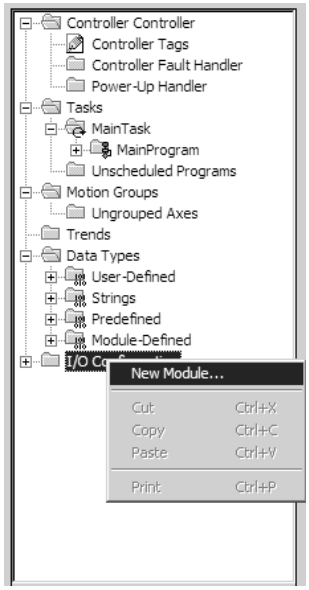
1 Open the Sample Ladder Logic in RSLogix 5000.
2 Start another instance of RSLogix 5000, and then open your existing application.

You will be adding the MVI56E-MCM module definition, and then copying controller tags, ladder logic, and user defined data types from the sample application into your existing application.

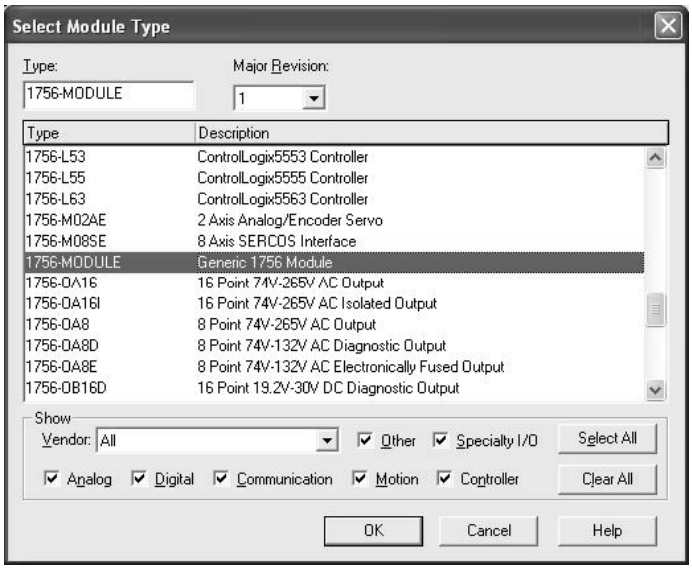
Defining Module in I/O Configuration

Note: You cannot perform this procedure while you are online to the controller.

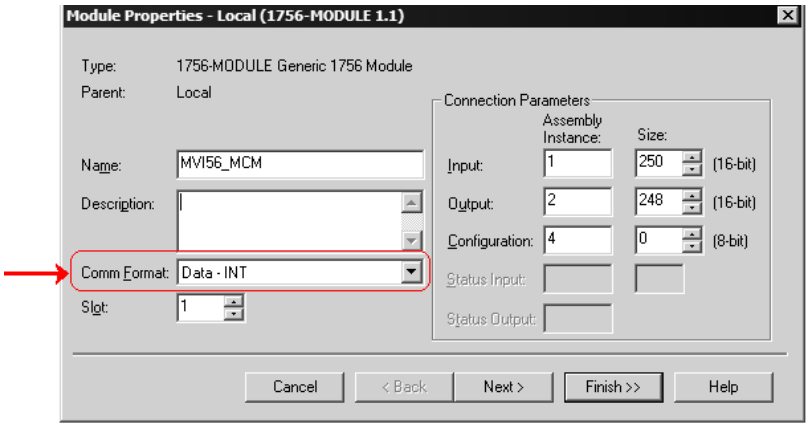
1 In the **CONTROLLER ORGANIZATION** list in RSLogix 5000, click the right mouse button on the **I/O CONFIGURATION** icon to open a shortcut menu. On the shortcut menu, choose **NEW MODULE…**. This action opens the **SELECT MODULE TYPE** dialog box.



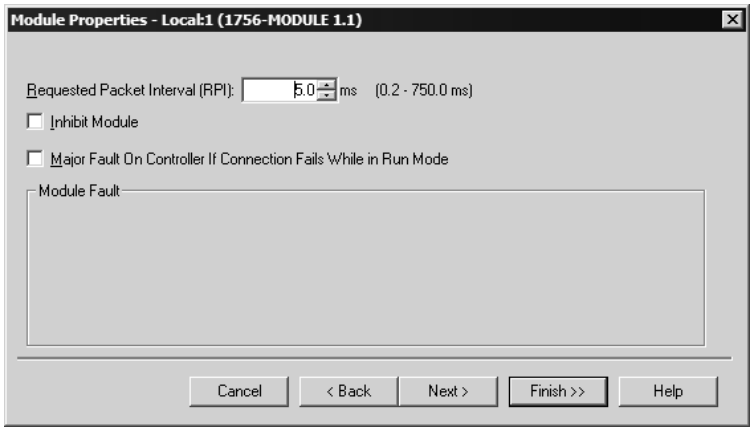
2 In the **SELECT MODULE TYPE** dialog box, select **1756-MODULE (GENERIC 1756 MODULE)** from the list and, and then click **OK.** This action opens the **MODULE PROPERTIES** dialog box.



3 In the **MODULE PROPERTIES** dialog box, enter the Name, Description and Slot options for your application, using the examples in the following illustration. You must select the Comm Format as **DATA - INT** in the dialog box. Failure to set the correct parameters will result in backplane communication problems between the module and the processor.



4 Click the **NEXT** button and set the Request Packet Interval to 5.0ms as shown in the following illustration.



5 Click **FINISH** to save the module into your existing application.

Copying the User Defined Data Types

Next, copy the User Defined Data Types from the sample program to your existing program. These data types contain configuration information, status, commands and other functions used by the program.

1 Arrange the two RSLogix 5000 windows on your desktop so that they are side-by-side.
2 In the **CONTROLLER ORGANIZATION** pane in the Sample Program, expand the **DATA TYPES** folder until the list of User-Defined data types is visible.
3 In the Sample Program window, select one data type at a time, and then drag the data type to the User-Defined data types folder in your existing program.

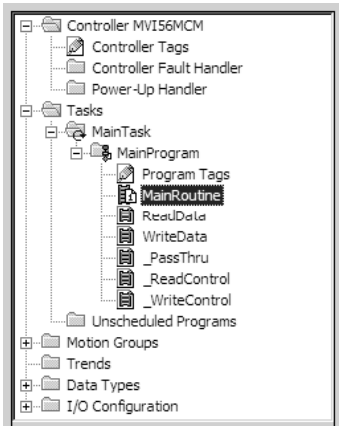
4 Repeat these steps until you have copied all of the data types from the sample program into your existing application.

Note: Data types prefixed with an underscore **[_]** are used in optional routines, and need not be copied unless your application requires them. Refer to MVI56E-MCM MVI56E-MCM User Defined Data Types (page 154) for a description of the usage for each data type.

Copying the Sample Ladder Logic

Next, copy the Sample Ladder Logic from the sample program to your existing program.

1 In the **CONTROLLER ORGANIZATION** pane in the Sample Program, expand the **TASKS** folder until the list of program routines is visible.



2 In the Sample Program window, select one routine at a time, and then drag the routine to the MainProgram folder in your existing program.
3 Save your program.

The sample program contains the following tasks.

MainRoutine

The **MAINROUTINE** checks for the presence of new read data from the module for the processor. The module cycles through its list of read blocks to transfer data from the module to the processor. Whenever new data is available, the module will set the value for the block in the module's input image **(LOCAL:1:I:DATA[249]).** The ladder logic must constantly scan this input word for a new value. The ladder logic should only perform the **READDATA** and **WRITEDATA** tasks, in that order, when a new value is present in **LOCAL:1:I:DATA[249],** otherwise data may be lost or scrambled.

If the new data is available, the **LASTREAD** and word (249) will not be equal. This will force the program to call the **READDATA** subroutine to process the new data received. After the new data is placed in the Modbus Data Table, the program will send new data to the module using the **WRITEDATA** subroutine.

ReadData

The **READDATA** task handles all new data and status information received from the module and placing it in the proper location in the processor. Data is transferred from the module to the processor using the module's input image **(LOCAL:1:I:DATA[ ]).** This task should set the last read block number (MCM1.BP.LastRead) to the current block number sent from the module **(LOCAL:1:I:DATA[249])** and stores the newly received read block number **(DATA[249])** into the **LASTREAD** variable.

Note: The **_READCONTROL** routine handles the command control responses received from the module. If command control, event command, or Slave status blocks are not going to be used in the application, then the **_READCONTROL** rung (rung 4 in the sample **READDATA** task) and the **_READCONTROL** and **_WRITECONTROL** ladder files may be removed.

If the module is configured for zero blocks, it will send blocks with identification codes of zero and -1. These blocks will only contain status data, and no user data will be included in these blocks.

The ladder obtains status information when the module is configured for either 1 or 0 blocks of read data. If the module is configured with 0 for the ReadRegCnt, then blocks -1 and 0 will be given by the module on the input image. If the ReadRegCnt is 200 or less, then you will receive block 0 and block 1.

The ladder logic also determines if the new data received in the input image is user data. If user data is present, the ladder logic will place the data in the correct location in the processor's read data area **(MCM.READDATA[ ])**. Up to 200 data words can be transferred in each block transfer. In addition to the user data, the block also contains important status data. This data should be copied to the correct data area in the module **(MCM.STATUS).** This status data can be used to determine the "health" of the MVI56E-MCM module. This rung computes offset into the Modbus Data Table for the received data block and to store the data into the Modbus Data Table.

If the requested block is within the valid range of data blocks for the Modbus Data Table, the offset into the table is computed as (Block ID number - 1) * 200. This is the starting offset in the Modbus Data Table where the 200 bits of data will be stored.

When the processor receives a pass-through block the received data will be handled at the _Pass-Thru routine. If the module is being used as a Modbus Master (PortX.Type=0) or a standard Modbus Slave (Port X.Type = 1) then this rung of logic and the _PassThru routine are not required. If the module is being used as PortX.Type = 2 to 4, then this rung and ladder routine is required.

WriteData

The WriteData task sends data from the processor to the MVI56E-MCM module. Data is transferred from the processor to the module using the module's output image **(LOCAL:1:O:DATA[ ]).** This task should store the currently requested data set in the module's **MCM.BP.LASTWRITE** data object. This object is used in all subsequent ladder logic in case the input word **(LOCAL:1:I:DATA[1])** changes during processing.

Note: The **_WRITECONTROL** routine handles the command control blocks sent to the module. If command control, event command, or Slave status blocks are not going to be used in the application, then the **_WRITECONTROL** rung (rung 7 in the sample **WRITEDATA** task) and the **_READCONTROL** and **_WRITECONTROL** ladder files may be removed.

_PassThru

Use this optional task to send pass-through data between the processor and the Modbus devices connected to the MVI56E-MCM module. Pass-Through functionality allows the Modbus Master to read and write the same Modbus address on a Modbus Slave. If pass-through mode is not chosen, then the attached Modbus Master device must read from one set of Modbus register/bits and write to another set of Modbus register/bits. Pass-Through mode takes a Modbus write command (Function Codes 5, 6, 15, and 16) and passes that to the ControlLogix processor. The pass-through ladder logic then parses that information and updates the **MCM.DATA.WRITEDATA** array with the new value that has been written by the Modbus Master.

_ReadControl

Use this optional task to get status and event data from the Modbus devices connected to the MVI56E-MCM module. Special command blocks requested from the module in the _WriteControl routine are processed and handled in this routine. If command control, event command, or Slave status blocks are not going to be used in the application, then this rung and the _ReadControl and _WriteControl ladder files may be removed.

_WriteControl

Use this optional task to send commands to the Modbus devices connected to the MVI56E-MCM module. Command Control, Event Command, and Slave status blocks are sent to the module in this task.

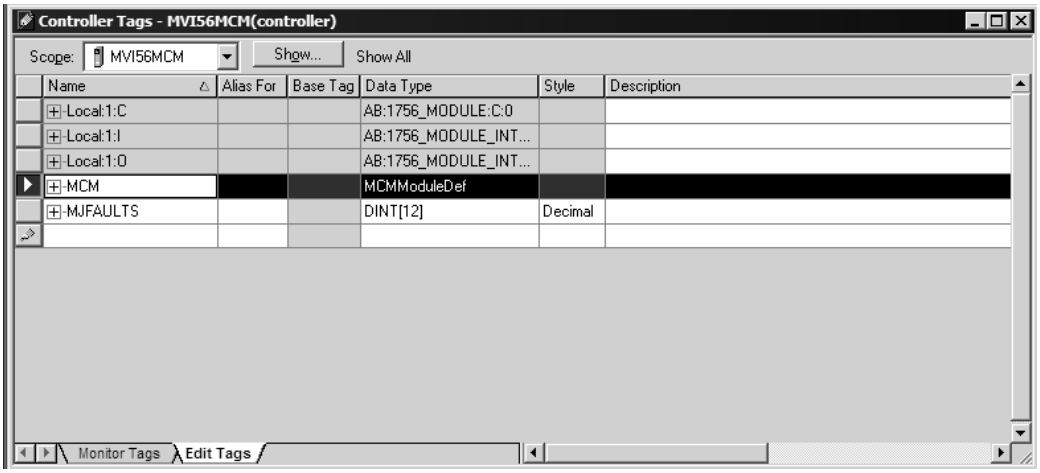
Copying the Controller Tags

Next, copy the Controller Tags from the sample program to your existing program. The sample program includes the following tags in the MCM structure.

- **MCM.MODDEF** configures the database in the module. The module uses this database to store input and output data transferred between the processor and the Modbus devices connected to the MVI56E-MCM module.
- **MCM.PORT1** and **MCM.PORT2** configure the module's serial ports for Modbus communications. The sample program configures Port 1 as a Modbus Master, and Port 2 as a Modbus Slave.
- **MCM.P1.CMD** and **MCM.P2.CMD** configure the Modbus Master commands for the module. These commands are active only if a port is configured as a Modbus Master.
- **MCM.READDATA** contains data read by the ControlLogix processor from the MVI56E-MCM module.
- **MCM.WRITEDATA** contains data read from the ControlLogix processor to the module's internal database.

The remaining controller tags contain error and status information, and special commands to execute.

1. In the **CONTROLLER ORGANIZATION** pane in each instance of RSLogix 5000, expand the **CONTROLLER** folder.
2. Double-click the **CONTROLLER TAGS** icon in each instance of RSLogix 5000. This action opens the **CONTROLLER TAGS** dialog box.
3. In the **CONTROLLER TAGS** dialog box in each instance of RSLogix 5000, click the **EDIT TAGS** tab, located at the bottom of the dialog box.
4. In the Sample Program, select the line containing the **MCM** tag structure.

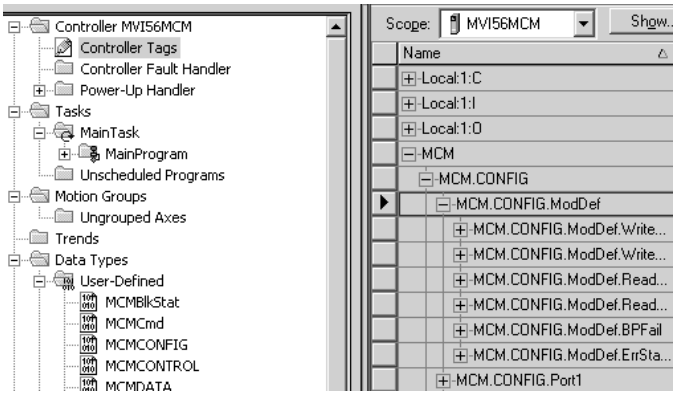


5. Drag the **MCM** tag structure to the blank line at the bottom of the list in the **EDIT TAGS** tab in your existing program.

Editing the Controller Tags

The MVI56E-MCM module configuration is stored in the **MCM.CONFIG** structure in the **CONTROLLER TAGS** dialog box. The sample program configures the module as a Modbus Master on Port 1, and a Modbus Slave on Port 2.

To edit the module configuration, in the Controller Organization pane, expand the **CONTROLLER** folder, and then double-click **CONTROLLER TAGS**. This action opens **CONTROLLER TAGS - MVI56MCM**, as shown in the following illustration.



- To configure a Modbus Master, refer to Configuration as a Modbus Master (page 33).
- To configure a Modbus Slave, refer to Configuration as a Modbus Slave (page 56).

Note: In order for any of the new values entered into these fields to be used by the module, you must restart the module (WarmBoot, ColdBoot, or cycle power).

7 Support, Service & Warranty

In This Chapter

- Contacting Technical Support 195
- Warranty Information 196

7.1 Contacting Technical Support

ProSoft Technology, Inc. is committed to providing the most efficient and effective support possible. Before calling, please gather the following information to assist in expediting this process:

1. Product Version Number
2. System architecture
3. Network details

If the issue is hardware related, we will also need information regarding:

1. Module configuration and associated ladder files, if any
2. Module operation and any unusual behavior
3. Configuration/Debug status information
4. LED patterns
5. Details about the serial, Ethernet or Fieldbus devices interfaced to the module, if any.

Note: *For technical support calls within the United States, ProSoft's 24/7 after-hours phone support is available for urgent plant-down issues. Detailed contact information for all our worldwide locations is available on the following page.*

Internet	Web Site: www.prosoft-technology.com/support E-mail address: support@prosoft-technology.com
Asia Pacific (location in Malaysia)	Tel: +603.7724.2080 E-mail: asiapc@prosoft-technology.com Languages spoken include: Chinese, English
Asia Pacific (location in China)	Tel: +86.21.5187.7337 x888 E-mail: asiapc@prosoft-technology.com Languages spoken include: Chinese, English
Europe (location in Toulouse, France)	Tel: +33 (0) 5.34.36.87.20 E-mail: support.EMEA@prosoft-technology.com Languages spoken include: French, English
Europe (location in Dubai, UAE)	Tel: +971-4-214-6911 E-mail: mea@prosoft-technology.com Languages spoken include: English, Hindi
North America (location in California)	Tel: +1.661.716.5100 E-mail: support@prosoft-technology.com Languages spoken include: English, Spanish
Latin America (Oficina Regional)	Tel: +1-281-2989109 E-Mail: latinam@prosoft-technology.com Languages spoken include: Spanish, English
Latin America (location in Puebla, Mexico)	Tel: +52-222-3-99-6565 E-mail: soporte@prosoft-technology.com Languages spoken include: Spanish
Brasil (location in Sao Paulo)	Tel: +55-11-5083-3776 E-mail: brasil@prosoft-technology.com Languages spoken include: Portuguese, English

7.2 Warranty Information

For complete details regarding ProSoft Technology's TERMS & CONDITIONS OF SALE, WARRANTY, SUPPORT, SERVICE AND RETURN MATERIAL AUTHORIZATION INSTRUCTIONS please see the documents on the Product DVD or go to www.prosoft-technology/legal

Documentation is subject to change without notice.

Index

_

_PassThru • 191
_ReadControl • 191
_WriteControl • 191

0

00 Return Query Data • 167

A

About the Modbus Protocol • 118
About the Optional Add-On Instruction • 16
Adding Multiple Modules (Optional) • 22
Adjust the Input and Output Array Sizes • 185
Adjust the Input and Output Array Sizes (Optional) • 27, 57
Assigning a Temporary IP Address • 83, 94
Assigning an IP Address in the Project • 81, 96

B

Backplane Configuration • 36, 103
Backplane Data Transfer • 118
Backplane Setup • 144
Backplane Status • 103
Battery Life Advisory • 4
Before You Begin • 20, 172
Before You Import the Add-On Instruction • 11, 15

C

Cable Connections • 138
Choosing the Controller Type • 183
Clearing a Fault Condition • 77
CmdControl • 158, 159
CoilArray • 160
Cold Boot Block (9999) • 132
Command Control • 53, 144, 152
Command Control and Event Command • 52
Command Control Blocks (5001 to 5006 or 5101 to 5106) • 125
Command Error Codes • 69, 72, 73, 106
Command List Entry Errors • 72
Commands Supported by the Module • 160
Configuration as a Modbus Master • 33, 193
Configuration as a Modbus Slave • 57, 193
Configuration Data Transfer • 126
Configuration Error Codes • 111
Configuring the Data Analyzer • 107
Configuring the RSLinx Driver for the PC COM Port • 31
Connect to the Module's Web Page • 16, 113
Connecting Your PC to the ControlLogix Processor • 28
Connecting Your PC to the Module • 83, 98
Connecting Your PC to the Module's Ethernet Port • 94
Contacting Technical Support • 195
Content Disclaimer • 2
Copying the Controller Tags • 191
Copying the Sample Ladder Logic • 189
Copying the User Defined Data Types • 188
Creating a New RSLogix 5000 Project • 14
Creating the Module • 16
Customizing the Memory Map • 61

D

Data Analyzer • 107
Data Analyzer Tips • 109
Data Flow Between MVI56E-MCM Module and ControlLogix Processor • 134
DB9 to RJ45 Adaptor (Cable 14) • 143
Defining Module in I/O Configuration • 186
Deployment Checklist • 10
Diagnostics (Function Code 08) • 167
Diagnostics and Troubleshooting • 67, 75, 76
Download the IP Address through CIPconnect • 83
Downloading the Project to the Module • 81, 98
Downloading the Sample Program to the Processor • 30, 185

E

Editing the Controller Tags • 192
Enron/Daniel Float Configuration • 64
Ethernet Cable Configuration • 139
Ethernet Cable Specifications • 138
Ethernet LED Indicators • 75
Ethernet Performance • 139
Event Command • 54
Event Command Blocks (1000 to 1255 or 2000 to 2255) • 122
EventCmd • 158, 159
Example 1
  Local Rack Application • 85
Example 2
  Remote Rack Application • 89
Example and State Diagram • 168

F

Floating-Point Data Handling (Modbus Master) • 41, 42, 46
Floating-Point Data Handling (Modbus Slave) • 62, 63
Force (Write) Multiple Coils 0x (Modbus Function Code 15) • 44
Force (Write) Single Coil 0x (Modbus Function Code 5) • 44
Force Multiple Coils (Function Code 15) • 169
Force Single Coil (Function Code 05) • 165
Formatted Pass-Through Blocks (9956 to 9959) • 130
Function 15 • 131

Function 5 • 130
Function 6 and 16 • 130
Functional Overview • 118
Functional Specifications • 116

G

General Specifications • 115
General Specifications - Modbus Master/Slave • 116

H

Hardware Specifications • 117

I

Import the Ladder Rung • 19
Important Safety Information • 3
Importing the Utility Add-On Instruction • 173
Installing ProSoft Configuration Builder • 78
Installing the Module in the Rack • 13

M

MainRoutine • 189
Master Command Configuration • 34, 37, 150
Master Command Data List (6000 to 6012 or 6100 to 6112) • 128
Master Command Examples • 41
Master Command List • 106, 138
Master Command Status • 106
Master Driver • 136
MCM Status Data • 72
MCMBlkStat • 157, 158
MCMCmd • 154, 156
MCMCONFIG • 154
MCMCONTROL • 154, 158, 160
MCMDATA • 154, 157
MCMModule • 154, 155
MCMModuleDef • 154
MCMPort • 106, 154, 155
MCMPortErrors • 157
MCMSTATUS • 154, 157
Misc. Status • 105, 150
Modbus Exception Codes • 171
Modbus Exception Responses • 170
Modbus Memory Map • 59, 61, 119
Modbus Protocol Specification • 160
ModDef Settings • 34, 58, 103
Module Communication Error Codes • 71
Module Configuration Block (9000) • 127
Monitoring Backplane Information • 103
Monitoring Database Information • 104
Monitoring General Information • 105
Monitoring Modbus Port Information • 105, 111
MVI56E-MCM Configuration Data • 144, 154
MVI56E-MCM Database Definition • 104, 143
MVI56E-MCM Remote Master Control • 132
MVI56E-MCM Status Data Definition • 67, 104, 111, 144, 152
MVI56E-MCM Status Data Definition as a Master • 67
MVI56E-MCM Status Data Definition as a Slave • 73
MVI56E-MCM User Defined Data Types • 154, 189

N

Non-Scrolling LED Status Indicators • 77
Normal Data Transfer • 120

O

Opening the Sample Program in RSLogix • 181
Other Modbus Addressing Schemes • 40
Overview • 33, 57, 172

P

Package Contents • 12
Pass-Through Blocks • 62, 129
Passthru • 158, 160
Pinouts • 70, 71, 115, 117, 138, 139, 143
Port 1 Commands • 150
Port 1 Setup • 145
Port 2 Commands • 150
Port 2 Setup • 147
Port Configuration • 35, 106
Port Status • 106
Preset (Write) Multiple Registers 4x (Modbus Function Code 16) • 46
Preset (Write) Single Register 4x (Modbus Function Code 6) • 45
Preset Multiple Registers (Function Code 16) • 170
Preset Single Register (Function Code 06) • 166
Product Specifications • 115

R

Read and Write Same Modbus Address (Pass Through) • 65
Read Block • 120
Read Coil Status (Function Code 01) • 161
Read Coil Status 0x (Modbus Function Code 1) • 42
Read Floating-Point Data • 47
Read Floats with Single Modbus Register Address (Enron/Daniel Float) • 50
Read Holding Registers (Function Code 03) • 163
Read Holding Registers 4x (Modbus Function Code 3) • 41
Read Input Registers (Function Code 04) • 164
Read Input Registers 3x (Modbus Function Code 4) • 42
Read Input Status (Function Code 02) • 162
Read Input Status 1x (Modbus Function Code 2) • 43
Read Multiple Floating-Point Registers • 48, 50
ReadData • 190
Reading Status Data from the Module • 111
Reading the Clock Value from the Module • 179
Reading the Ethernet Settings from the Module • 176
Reference • 115
RS-232
  Modem Connection (Hardware Handshaking Required) • 140
  Null Modem Connection (Hardware Handshaking) • 141

Null Modem Connection (No Hardware Handshaking) • 141
RS-232 Application Port(s) • 140
RS-422 • 142
RS-485 and RS-422 Tip • 143
RS-485 Application Port(s) • 142

S

Scrolling LED Status Indicators • 76
Selecting the Slot Number for the Module • 184
Setting Jumpers • 12
Setting Up ProSoft Configuration Builder • 78
Setting Up the Project • 79
Slave Configuration • 62
Slave Driver • 135
Slave Status Blocks (3000 to 3003 or 3100 to 3103) • 106, 123
Slave Status List • 106
SlavePollStatus • 158, 159
Special Function Blocks • 122
Standard Modbus Protocol Errors • 70
Start Here • 9
Starting the Data Analyzer • 107
Stopping the Data Analyzer • 108
Sub-function Codes Supported • 167
Support, Service & Warranty • 195
System Requirements • 9

T

The Diagnostics Menu • 103
Transferring the Command Error List to the Processor • 72
Troubleshooting the LEDs • 78

U

Unformatted Pass-Through Blocks (9996) • 129
Using RSWho to Connect to the Module • 92
Using the Diagnostics Menu in ProSoft Configuration Builder • 99
Using the Optional Add-On Instruction • 16, 172
Using the Sample Program - RSLogix 5000 Version 15 and earlier • 11, 15, 180
Using the Sample Program in a New Application • 181
Using the Sample Program in an Existing Application • 120, 126, 135, 137, 186
Util • 154, 158

V

Verify Communication • 36, 67
Verify Slave Communications • 73
Verifying Master Communications • 67
Viewing the Error Status Table • 111

W

Warm Boot Block (9998) • 132
Warranty Information • 196
Write Block • 121
Write Configuration Block (-9000 and -6000 to -6003 or -6100 to -6103) • 133
Write Floats to Slave Device • 49
Write to Enron/Daniel Floats • 51
WriteData • 190
Writing the Clock Value to the Module • 179
Writing the Ethernet Settings to the Module • 177

Y

Your Feedback Please • 2